



Rust 编程之道

张汉东◎著



中国工信出版集团



电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
<http://www.phei.com.cn>

□□□□

Rust is a systems programming language that runs fast and is safe, secure, and easy to use. It is designed to be a good replacement for C/C++ in systems programming.

Rust
 Rust
 Rust
 Rust

Rust 2018 Rust Rust
 Rust Rust Rust Rust Rust

[illegible]

□□□□□(CIP)□□

Rust 1.0/1.0.—November 2019.1

ISBN 978-7-121-35485-4

I.①R... II.□□... III.□□□□□-□□□□ IV.①TP312

□□□□□□CIP□□□□(2018)□251334□

--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--

111

111

[illegible]

□□□□□□□□173□□ □□□100036

□□□787×1092 1/16 □□□36.25 □□□1018□□

□□□2019□1□□1□

□□□2019□1□□1□□□

□□□128.00□

[illegible]

zlts@phei.com.cn
dbqq@phei.com.cn

☎☎☎☎☎☎☎(010)51260888-819✉faq@phei.com.cn

□□□□

Even though I had to read this book through Google Translate, *The Tao of Rust* is an extremely interesting book. It starts off explaining exactly why it is different: it's a book that gets you to think about Rust, and its perspective on the world. I only wish I could read it in its native tongue, as I'm sure it's even better than! I have been working on Rust for six years now, and this book changed my perspective on some aspects of the language. That's very powerful!

□□□□□□□□□□□□□□□□□□□□ Rust □□□□□□□□□□□□□□
□ Rust □□□□□□□□□□□□□□ Rust □□ Rust □□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□ Rust □□□□□□□□□□□□□□ Rust □□□□
□□□□□□□□

——Steve Klabnik □ Rust □□□□□□□□□□□□□□



I knew Rust was a notoriously difficult programming language to learn, but it wasn't until I read the preface to *The Tao of Rust*, by Alex Zhang, that I realized why it is so difficult. Alex writes:

Rust covers a wide range of knowledge, including object-oriented, functional programming, generics, underlying memory management, type systems, design patterns, and more.

Alex covers all of these topics and more in *The Tao of Rust*. A single text that ties all of this together will be invaluable for Rust learners. So far I've read a couple of chapters translated from the original Chinese, and I can't wait to read more.

Rust is a systems programming language that Alex Zhang wrote. Alex Zhang is a Rust enthusiast.

"Rust is a systems programming language that Alex Zhang wrote. Alex Zhang is a Rust enthusiast."

Rust is a systems programming language that Alex Zhang wrote. Rust is a systems programming language that Alex Zhang wrote.

—Patrick Shaughnessy, Ruby enthusiast

--	--	--	--

TiDB key-value TiKV
Go C++ Java Rust
Rust Rust

- 内存碎片太多，导致无法申请大空间
- 解决方法：
- 使用垃圾回收（GC）来管理内存
 - 手动释放不再使用的变量

Rust Rust TiKV
 TiKV CNCF Cloud

[illegible]

Rust C++ Rust Rust Rust Rust Rust

□□□□ Rust□□□□□□

——PingCAP TiKV——



2015 Rust Rust

Ruby 2006

C++

C++

Rust 1.0 Rust

-
- GC
- LLVM
-
- +
-
-
-

Rust

Logo Logo

Logo

C

Ruby Java Java

Web Ruby on Rails

Ruby “ ”

Ruby 的 庫 函 數 庫 是 在 2012 年 12 月 1 日 發 布 的 ， 這 是 Ruby 2.0.0 的 一 個 重 大 更 新 。

Ruby 的 庫 函 數 庫 是 在 2012 年 12 月 1 日 發 布 的 ， 這 是 Ruby 2.0.0 的 一 個 重 大 更 新 。

Ruby

在 2012 年 12 月 1 日 發 布 的 Ruby 2.0.0 的 一 個 重 大 更 新 。

- 庫 函 數 庫
- 庫 函 數 庫
- 庫 函 數 庫

在 2012 年 12 月 1 日 發 布 的 Ruby 2.0.0 的 一 個 重 大 更 新 。

Ruby 的 庫 函 數 庫 是 在 2012 年 12 月 1 日 發 布 的 ， 這 是 Ruby 2.0.0 的 一 個 重 大 更 新 。

Haskell 的 庫 函 數 庫 是 在 2012 年 12 月 1 日 發 布 的 ， 這 是 Haskell 2.0.0 的 一 個 重 大 更 新 。

Rust 的 庫 函 數 庫 是 在 2012 年 12 月 1 日 發 布 的 ， 這 是 Rust 2.0.0 的 一 個 重 大 更 新 。

Rust 的 庫 函 數 庫 是 在 2012 年 12 月 1 日 發 布 的 ， 這 是 Rust 2.0.0 的 一 個 重 大 更 新 。

Rust

序

本书是 Rust 的入门书籍，书名“Rust 入门”已经说明了本书的定位。本书旨在帮助读者了解 Rust 语言，并掌握其基本语法和编程思想。本书适合初学者阅读，也适合有一定编程经验的读者作为参考。

本书从 Rust 语言的基本概念入手，介绍了 Rust 的编译原理、内存管理、并发编程等核心特性。通过大量的代码示例，读者可以直观地理解 Rust 的语法和编程思想。本书还介绍了 Rust 的生态系统，包括标准库、第三方库和开发工具等。

本书是 Rust 语言的入门书籍，旨在帮助读者了解 Rust 语言的基本概念和编程思想。本书适合初学者阅读，也适合有一定编程经验的读者作为参考。

Rust 语言的入门书籍 *Rust Book* 是 Rust 语言的官方入门书籍，也是目前最权威的 Rust 入门书籍。本书详细介绍了 Rust 语言的基本概念和编程思想，适合初学者阅读。此外，还有 *Rust Book* 2 和 *Rust Book* 3 等书籍，分别介绍了 Rust 语言的进阶知识和应用。对于想要快速了解 Rust 语言的读者，*Rust Primer* 和 *Programming Rust* 也是不错的选择。

本书是 Rust 语言的入门书籍，旨在帮助读者了解 Rust 语言的基本概念和编程思想。本书适合初学者阅读，也适合有一定编程经验的读者作为参考。本书详细介绍了 Rust 语言的基本概念和编程思想，包括变量、函数、控制流、数据结构、并发编程等。通过大量的代码示例，读者可以直观地理解 Rust 的语法和编程思想。

本书是 Rust 语言的入门书籍，旨在帮助读者了解 Rust 语言的基本概念和编程思想。本书适合初学者阅读，也适合有一定编程经验的读者作为参考。本书详细介绍了 Rust 语言的基本概念和编程思想，包括变量、函数、控制流、数据结构、并发编程等。通过大量的代码示例，读者可以直观地理解 Rust 的语法和编程思想。本书还介绍了 Rust 的生态系统，包括标准库、第三方库和开发工具等。

Rust
 Rust
 Rust Rust
 Rust
 Rust

Rust
 Rust Rust
 Rust Rust
 Rust

Rust
 Rust Rust
 Rust Rust
 Rust

Rust
 Rust Rust
 Rust Rust
 Rust

Rust
 Rust Rust
 Rust Rust
 Rust

- Rust
- Rust Rust

Rust
 Rust

Rust
 Rust

Rust
 Rust
 Rust

Bug

[illegible]

- doc.rust-lang.org
- Rust [\[1\]](#) Rust Rust Rust

Mike Rust Primer Rust
 Rust Book Rust KiChjang
 ELTON CrLF0710 F001 Lingo tennix iovxw wayslog
 Xidorn 42 Rust
 Rust
 Rust

[illegible]

1111

www.broadview.com.cn

- 本书 全面系统地介绍了 Rust 语言
- 本书 全面系统地介绍了 Rust 语言

- 本书 全面系统地介绍了 Rust 语言

<http://www.broadview.com.cn/35485>



[\[1\] https://github.com/RustStudy/rust_daily_news](https://github.com/RustStudy/rust_daily_news)



[□□□□](#)

[□□□□](#)

[□□□□](#)

[□](#)

[□□](#)

[□1□ □□□□□□](#)

[1.1 □□](#)

[1.2 □□□□](#)

[1.2.1 □□□□](#)

[1.2.2 □□□□□](#)

[1.2.3 □□□](#)

[1.3 □□□□□](#)

[1.3.1 □□□□](#)

[1.3.2 □□□□](#)

[1.3.3 □□□□](#)

[1.4 Rust□□□□□□](#)

[1.5 □□](#)

[□2□ □□□□](#)

[2.1 Rust□□□□□□□](#)

[2.1.1 □□□□](#)

[2.1.2 □□□](#)

[2.1.3 □□□](#)

[2.1.4 □□□](#)

[2.1.5 □□□□](#)

[2.2 関数型言語](#)

[2.3 関数型言語](#)

[2.3.1 関数型言語の基礎](#)

[2.3.2 関数型言語の応用](#)

[2.3.3 関数型言語の発展](#)

[2.4 関数型言語](#)

[2.4.1 関数型言語](#)

[2.4.2 関数型言語の応用](#)

[2.4.3 関数型言語](#)

[2.4.5 CTFE関数](#)

[2.4.6 関数](#)

[2.5 関数型言語](#)

[2.5.1 関数型言語](#)

[2.5.2 関数型言語](#)

[2.5.3 match関数型言語](#)

[2.5.4 if letwhile let関数](#)

[2.6 関数型言語](#)

[2.6.1 関数型言語](#)

[2.6.2 関数型言語](#)

[2.6.3 関数型言語](#)

[2.6.4 関数型言語](#)

[2.6.5 関数型言語](#)

[2.6.6 関数型言語](#)

[2.6.7 str関数型言語](#)

[2.6.8 関数型言語](#)

[2.6.9 never関数](#)

[2.7 関数型言語](#)

[2.7.1 関数](#)

[2.7.2](#) [□□□](#)

[2.7.3](#) [□□□](#)

[2.8](#) [□□□□□□](#)

[2.8.1](#) [□□□□□□□](#)

[2.8.2](#) [□□□□□□□□□](#)

[2.8.3](#) [□□□□□□□](#)

[2.8.4](#) [Key-Value□□□□HashMap□BTreeMap](#)

[2.8.5](#) [□□□HashSet□BTreeSet](#)

[2.8.6](#) [□□□□□BinaryHeap](#)

[2.9](#) [□□□□](#)

[2.10](#) [□□□trait](#)

[2.10.1](#) [□□](#)

[2.10.2](#) [trait](#)

[2.11](#) [□□□□](#)

[2.12](#) [□□□□□□](#)

[2.13](#) [□□□□□□](#)

[2.14](#) [□□](#)

[□3□](#) [□□□□](#)

[3.1](#) [□□□□](#)

[3.1.1](#) [□□□□□□□](#)

[3.1.2](#) [□□□□□□□](#)

[3.1.3](#) [□□□□□□□□](#)

[3.2](#) [Rust□□□□□□](#)

[3.2.1](#) [□□□□](#)

[3.2.2](#) [□□□□](#)

[3.3](#) [□□](#)

[3.3.1](#) [□□□□](#)

[3.3.2](#) [□□□□□□□□□](#)

[3.4 trait](#)

[3.4.1](#)

[3.4.2](#)

[3.4.3](#)

[3.4.4 trait](#)

[3.5](#)

[3.5.1 Deref](#)

[3.5.2 as](#)

[3.5.3 FromInto](#)

[3.6 trait](#)

[3.6.1](#)

[3.6.2](#)

[3.6.3](#)

[3.7](#)

[4](#)

[4.1](#)

[4.1.1](#)

[4.1.2](#)

[4.1.3](#)

[4.2 Rust](#)

[4.2.1](#)

[4.2.2 RAI](#)

[4.2.3](#)

[4.2.4](#)

[4.3](#)

[5](#)

[5.1](#)

[5.2](#)

[5.3 `Vec` の実装](#)

[5.3.1 `Vec` の構造](#)

[5.3.2 `Vec` の静的メンバ関数の実装](#)

[5.4 `Vec` のテスト](#)

[5.5 `Vec` の応用](#)

[5.5.1 `Vec` の応用](#)

[5.5.2 `Vec` の応用](#)

[5.5.3 `Vec` の応用](#)

[5.5.4 `Vec` の応用](#)

[5.6 `Vec` の応用](#)

[5.6.1 `Vec` の応用](#)

[5.6.2 `Vec` の応用](#)

[5.6.3 `Vec` の応用](#)

[5.7 `Vec` の応用](#)

[5.8 `Vec` の応用](#)

[5.9 `Vec`](#)

[第6章 `Vec` の応用](#)

[6.1 `Vec`](#)

[6.1.1 `Vec`](#)

[6.1.2 `Vec`](#)

[6.1.3 `Vec`](#)

[6.1.4 `Vec`](#)

[6.1.5 `Vec`](#)

[6.1.6 `Vec`](#)

[6.2 `Vec`](#)

[6.2.1 `Vec`](#)

[6.2.2 `Vec`](#)

[6.2.3 `Vec`](#)

[6.2.4 遍历器（Iterator）](#)

[6.2.5 遍历器对象的遍历](#)

[6.3 解构](#)

[6.3.1 数组解构赋值](#)

[6.3.2 Iterator trait](#)

[6.3.3 IntoIterator trait 和遍历器](#)

[6.3.4 遍历器对象的遍历](#)

[6.3.5 遍历器对象的遍历](#)

[6.3.6 遍历器对象的遍历](#)

[6.4 其他](#)

[第7章 函数](#)

[7.1 函数](#)

[7.1.1 函数](#)

[7.1.2 函数](#)

[7.1.3 函数](#)

[7.2 函数](#)

[7.2.1 函数](#)

[7.2.2 函数](#)

[7.2.3 RAII 模式](#)

[7.3 其他](#)

[第8章 其他](#)

[8.1 其他](#)

[8.1.1 其他](#)

[8.1.2 其他](#)

[8.1.3 其他](#)

[8.1.4 其他](#)

[8.1.5 其他](#)

[8.1.6 其他](#)

[8.1.7 関数型プログラミング](#)

[8.1.8 関数型プログラミング](#)

[8.2 関数型プログラミング](#)

[8.2.1 関数型プログラミング](#)

[8.2.2 関数型プログラミング](#)

[8.3 関数型プログラミング](#)

[8.4 関数型プログラミング](#)

[9 関数型プログラミング](#)

[9.1 関数型プログラミング](#)

[9.2 関数型プログラミング](#)

[9.3 関数型プログラミング](#)

[9.3.1 関数型Option型T](#)

[9.3.2 関数型Result型T E](#)

[9.4 関数型Panic](#)

[9.5 関数型プログラミング](#)

[9.6 関数型プログラミング](#)

[10 関数型プログラミング](#)

[10.1 関数型プログラミング](#)

[10.1.1 関数Cargo関数](#)

[10.1.2 関数型プログラミング](#)

[10.1.3 Cargo.toml関数型](#)

[10.1.4 関数Cargo](#)

[10.2 関数型プログラミング](#)

[10.3 関数型プログラミング](#)

[10.3.1 関数Cargo関数型](#)

[10.3.2 関数structopt関数型](#)

[10.3.3 関数型プログラミング](#)

[10.3.4 関数CSV関数](#)

[10.3.5 `CSV` フォーマット](#)

[10.3.6 `CSV` の読み込み](#)

[10.4 `CSV` の書き込み](#)

[10.5 `CSV` の読み込み](#)

[第11章 `Parallelism`](#)

[11.1 `Parallelism` の基礎](#)

[11.1.1 `Parallelism` の基礎](#)

[11.1.2 `Parallelism` の基礎](#)

[11.1.3 `Parallelism` の基礎](#)

[11.2 `Parallelism` の応用](#)

[11.2.1 `Parallelism` の応用](#)

[11.2.2 `SendSync`](#)

[11.2.3 `Parallelism` の応用](#)

[11.2.4 `Parallelism` の応用](#)

[11.2.5 `Parallelism` の応用](#)

[11.2.6 `Channel` の使用](#)

[11.2.7 `Parallelism` の応用](#)

[11.2.8 `Parallelism` の応用](#)

[11.2.9 `Rayon` の使用](#)

[11.2.10 `Crossbeam` の使用](#)

[11.3 `Parallelism` の応用](#)

[11.3.1 `Parallelism` の応用](#)

[11.3.2 `Future` の使用](#)

[11.3.3 `async/await`](#)

[11.4 `Parallelism` の応用](#)

[11.4.1 `SIMD` の使用](#)

[11.4.2 `Rust` の `SIMD`](#)

[11.5 `Parallelism` の応用](#)

[第12章 本章](#)

[12.1 本章](#)

[12.1.1 本章is本章](#)

[12.1.2 本章](#)

[12.1.3 本章](#)

[12.2 本章](#)

[12.2.1 本章](#)

[12.2.2 Rust本章](#)

[12.2.3 本章](#)

[12.2.4 本章](#)

[12.2.5 本章](#)

[12.3 本章](#)

[12.4 本章](#)

[第13章 本章](#)

[13.1 Unsafe Rust本章](#)

[13.1.1 Unsafe本章](#)

[13.1.2 本章](#)

[13.1.3 Union本章](#)

[13.1.4 本章](#)

[13.2 本章Unsafe本章](#)

[13.2.1 本章](#)

[13.2.2 本章](#)

[13.2.3 本章](#)

[13.2.4 Drop本章](#)

[13.2.5 NonNull本章T本章](#)

[13.2.6 Unsafe本章](#)

[13.2.7 本章](#)

[13.2.8 本章](#)

13.3 [□□□□□□](#)

13.3.1 [□□□□□](#)

13.3.2 [□C/C++□□□](#)

13.3.3 [□□Rust□□□□□□□](#)

13.4 [Rust□WebAssembly](#)

13.4.1 [WebAssembly□□□](#)

13.4.2 [□□Rust□□WebAssembly](#)

13.4.3 [□□WebAssembly□□□](#)

13.5 [□□](#)

[□□A Rust□□□□□](#)

[□□B Rust□□□□□](#)

第1章 简介

本章主要介绍 Rust 语言

Rust 语言是一种系统编程语言，它结合了 C 语言的性能和 Python 语言的易用性。Rust 语言由 Mozilla 基金会开发，旨在提供一种安全、快速、灵活的编程方式。Rust 语言的设计目标是提供一种安全、快速、灵活的编程方式，同时保持与 C 语言的性能相当。Rust 语言的设计目标是提供一种安全、快速、灵活的编程方式，同时保持与 C 语言的性能相当。

Rust 语言的设计目标是提供一种安全、快速、灵活的编程方式，同时保持与 C 语言的性能相当。Rust 语言的设计目标是提供一种安全、快速、灵活的编程方式，同时保持与 C 语言的性能相当。Rust 语言的设计目标是提供一种安全、快速、灵活的编程方式，同时保持与 C 语言的性能相当。Rust 语言的设计目标是提供一种安全、快速、灵活的编程方式，同时保持与 C 语言的性能相当。

Rust 语言的设计目标是提供一种安全、快速、灵活的编程方式，同时保持与 C 语言的性能相当。Rust 语言的设计目标是提供一种安全、快速、灵活的编程方式，同时保持与 C 语言的性能相当。Rust 语言的设计目标是提供一种安全、快速、灵活的编程方式，同时保持与 C 语言的性能相当。

1.1 背景

本章主要介绍 Rust 语言

Rust 语言的设计目标是提供一种安全、快速、灵活的编程方式，同时保持与 C++ 语言的性能相当。Rust 语言的设计目标是提供一种安全、快速、灵活的编程方式，同时保持与 C++ 语言的性能相当。Rust 语言的设计目标是提供一种安全、快速、灵活的编程方式，同时保持与 C++ 语言的性能相当。

- 快速
- 安全

Rust 语言的设计目标是提供一种安全、快速、灵活的编程方式，同时保持与 C/C++ 语言的性能相当。Rust 语言的设计目标是提供一种安全、快速、灵活的编程方式，同时保持与 C/C++ 语言的性能相当。Rust 语言的设计目标是提供一种安全、快速、灵活的编程方式，同时保持与 C/C++ 语言的性能相当。Rust 语言的设计目标是提供一种安全、快速、灵活的编程方式，同时保持与 C/C++ 语言的性能相当。

Rust 语言的设计目标是提供一种安全、快速、灵活的编程方式，同时保持与 C/C++ 语言的性能相当。Rust 语言的设计目标是提供一种安全、快速、灵活的编程方式，同时保持与 C/C++ 语言的性能相当。Rust 语言的设计目标是提供一种安全、快速、灵活的编程方式，同时保持与 C/C++ 语言的性能相当。

“**GH**”
Rust

“**Rust**”
Rust Fungi
“”
55
“” Rust Logo 1-1
Logo 55 Rust “Rust”
“” “” Rust
“Rust” “Trust” “Robust” “” “”
“Rust” Rust



1-1 Rust Logo




GH
20
Ada
[1]

GH
•
•
•
C/C++
GH Rust

GH [\[1\]](#)——[\[2\]](#)
[\[3\]](#)“[\[4\]](#)”[\[5\]](#)Rust[\[6\]](#)

1.2 实验目的

□□□□□□ Rust □□□□□□□□□□□□

- 
- 
- 

□□□□Rust□□□□□□□□□□□□□□□□□□□□□□□□Rust□□□□□□□□

1.2.1 实验目的

rust-rust-rust

□□□□□□□□“□□□□□□”□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□

- □□□□□□□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□□□□□□□□□

[illegible]

Rust

OCaml Haskell
“ ” Rust
Haskell Rust Haskell

Haskell Haskell
 Haskell
 Rust
 Rust

-
-
-
-
-

Rust
 Rust

•

•

C++RAII Rust GC

Rust Haskell

-
-
-
-
-
-
-
- trait

□□□□□□□□ Rust □□□□□□□□□□

- `Affine Type` `Rust` `Move`

- | | | | | | | | |
|--|--|--|--|--|--|--|--|
| | | | | | | | |
|--|--|--|--|--|--|--|--|

Rust

```

    BugBug
    RustRust
    Rust

```

1.2.2 实验目的

□□□□□ Rust □□□□□□□□□□

Ruby Ruby Rust 1-1

1-1 Ruby Rust

```
1. # Ruby 代码
2. 5.times{ puts "Hello Ruby" }

2.days.from_now
// Rust 代码
5.times(|| println!("Hello Rust"));
2.days().from now();
```

[illegible]

```

    Ruby
    Rust
    Rust
    1-1
    5
    Rust

```

Rust trait “”

1.2.3 概要

この章では、Rustの概要を説明します。

- ・ Rustの目的と特徴
- ・ Rustのインストールと実行方法
- ・ Rustの基本的な文法と型システム

Rustのインストールと実行方法

Rust

Rustのインストールと実行方法

Rustのインストールと実行方法

Rustのインストールと実行方法

Rustのインストールと実行方法

Rustのインストールと実行方法

- ・ Rustの基本的な文法と型システム
- ・ Rustの基本的な文法と型システム
- ・ Rustの基本的な文法と型システム

Rustのインストールと実行方法

Rustのインストールと実行方法

Unsafe Rust 的 unsafe 和 Safe Rust 的 Safe Rust 的 Unsafe Rust 的

Bug Rust 的 Safe Rust 的 Unsafe Rust 的 Bug 的

Rust 的 Cargo Rust 的 crate Cargo Rustfmt clippy 的 rustfix 的 Cargo Git crates.io 的

Rust Rust 的

- Rust Book [2] nomicon book Rust Bridge 的

- Rust Markdown Rust 的

- Playground Rust Rust Rust NLL 的

- Haskell Rust 的 Rust 的

- Rust 的

- Rust 的

Rustは、C/C++、RAII、C++、Haskell、GC、Java、Ruby、Pythonと比べて、

- ・ Rustは、C/C++、RAII、C++、Haskell、GC、Java、Ruby、Pythonと比べて、
- ・ Rustは、C/C++、RAII、C++、Haskell、GC、Java、Ruby、Pythonと比べて、
- ・ Rustは、C/C++、RAII、C++、Haskell、GC、Java、Ruby、Pythonと比べて、

Rustは、C/C++、RAII、C++、Haskell、GC、Java、Ruby、Pythonと比べて、

Rustは、C/C++、RAII、C++、Haskell、GC、Java、Ruby、Pythonと比べて、

1.3 Rust

2015年Rust 1.0は、Rustは、C/C++、RAII、C++、Haskell、GC、Java、Ruby、Pythonと比べて、

edition

- **Rust 2015** Rust 1.0~1.30

- **Rust 2018** Rust 1.31~Rust 2018

Rust try Rust "edition=2018" Rust try Rust

Rust

- Rust 2015~Rust 2018

- Rust javac Java 9~Java 10 gcc clang C++14~C++17

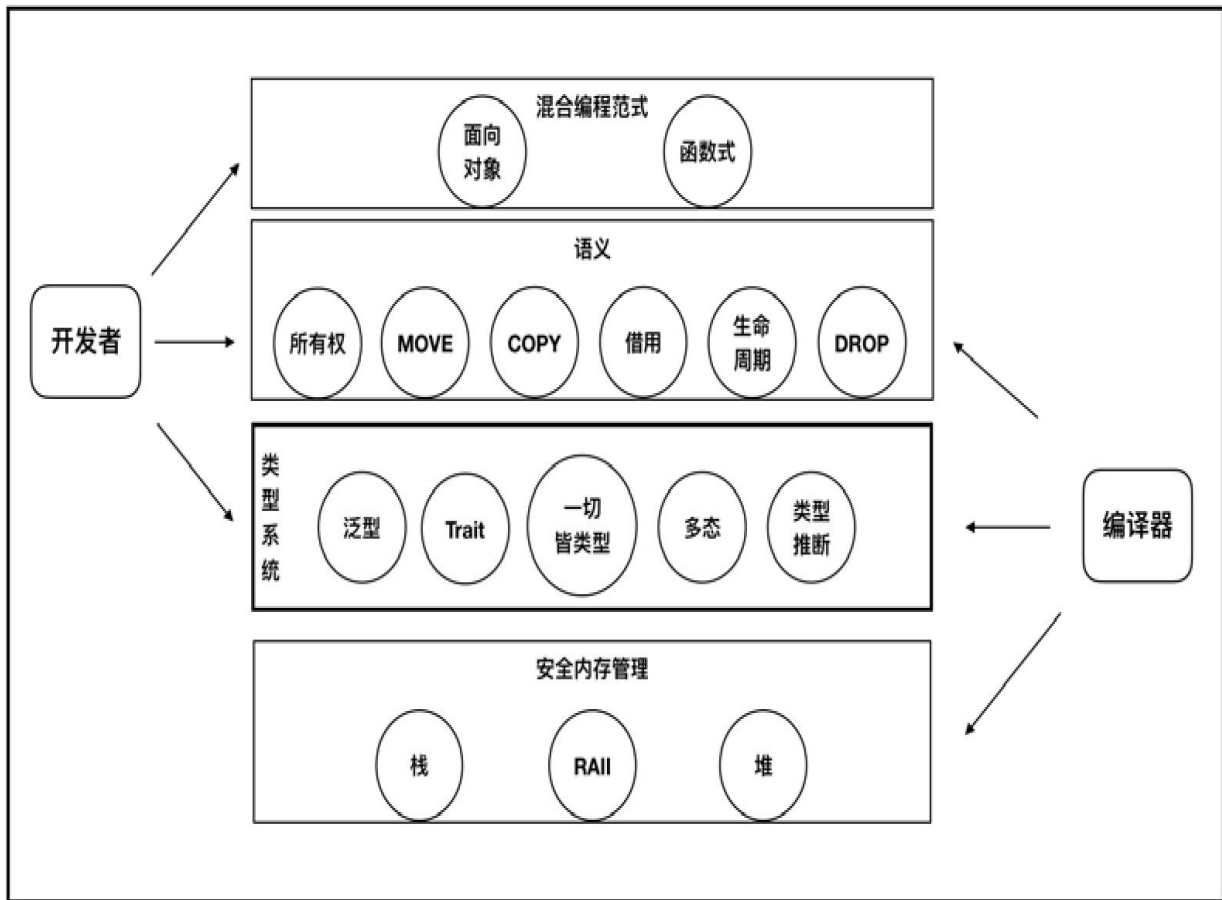
- Rust 2018~Rust 2015

- Rust

2021 Rust

1.3.1

Rust 1-2



1-2 Rust 的组成

1-2 Rust 的组成 4 部分

编译时、运行时、链接时、部署时

编译时：编译成目标代码、链接成可执行文件、生成库文件

Rust 的编译时、运行时、链接时、部署时的组成

编译时：编译成目标代码、链接成可执行文件、生成库文件

运行时：运行时的组成，包括“运行时”和“运行时库”

Rust 的运行时、链接时、部署时的组成

部署时

1.3.2 编译

Rust 的编译过程

```
    RustXXXXXXXXXXXXXXXXXXXXRustXXXXXXXXXXRustX
XXXXRustXXXXXXXXXXXXXXXXXXXXRustXXXXXXXXXXXX
```

Rust [3] Rust
1900 Rust Rust 2018 Rust
CLI WebAssembly

[RFC](#)
[RFC](#)
[Pull Request](#)

Rust ☐ **Stable** ☐ **Beta** ☐
☐ **Nightly** ☐ 6 ☐ **Unstable** ☐
☐ **Feature Gate** ☐

1.3.3 实验

Octoverse Rust PR 15 Rust 2604

□□□□□□ Rust □□□□□□□□□□□□□□□□□□

- Amazon  Rust 
- Atlassian  Rust 
- Dropbox  Rust 
- Facebook  Rust 
- Google  Fuchsia 
- Microsoft  Azure IoT 
- npm  Rust 
- RedHat  Rust 
- Reddit  Rust 
- Twitter  Rust 










Rust
 Mozilla Rust
 Rust Rust

1.4 Rust

```

    Rust
    Rust
    LLVM
    Rust
    LLVM
    LLVM IR
    LLVM

```

Rust  AST  AST  HIR
 High-level IR  HIR  MIR
 Middle IR  Rust1.12 

- 中间代码 MIR 是 LLVM 和 Rust 的接口
- 中间代码 MIR 是 LLVM 和 Rust 的接口
- 中间代码 MIR 是 LLVM 和 Rust 的接口

1.5 □□

Rust[2006]——
Rust
Rust“

[illegible]

[1] 2019年9月10日，Ada64编程语言在GitHub上发布了16个版本。

[2] *The Rust Programming Language*

[3] 2018年7月

第2章 環境構築

本章では、Rustの開発環境を構築します。Rustは、Linux、macOS、Windowsの3つのプラットフォームで動作します。本章では、LinuxとmacOSでの環境構築を説明します。Windowsでの環境構築については、Rustの公式ドキュメントを参照してください。

Rustの開発環境を構築するには、Rustのコンパイラと、Rustの標準ライブラリをインストールする必要があります。Rustのコンパイラは、Rustの公式ドキュメントに従ってインストールします。Rustの標準ライブラリは、Rustの公式ドキュメントに従ってインストールします。

2.1 Rustの開発環境

Rustの開発環境を構築するには、Rustのコンパイラと、Rustの標準ライブラリをインストールする必要があります。

- Rustのコンパイラ
- Rustの標準ライブラリ
- Rustのビルドシステム
- Rustのテストシステム
- Rustのデバッグシステム

2.1.1 Rustのインストール

Rustの開発環境を構築するには、Rustのコンパイラと、Rustの標準ライブラリをインストールする必要があります。Rustのコンパイラは、Rustの公式ドキュメントに従ってインストールします。Rustの標準ライブラリは、Rustの公式ドキュメントに従ってインストールします。

Rustのインストール

Rustの開発環境を構築するには、Rustのコンパイラと、Rustの標準ライブラリをインストールする必要があります。

- Rustのコンパイラ
- Rustの標準ライブラリ
- Rustのビルドシステム

编译选项 Rust 编译选项编译选项编译选项编译选项编译选项编译选项
Rust 编译选项编译选项编译选项编译选项编译选项编译选项编译选项

RFC

Rust 编译选项 RFC 编译选项 RFC 编译选项编译选项编译选项编译选项编译选项
编译选项编译选项编译选项 RFC 编译选项编译选项编译选项编译选项编译选项编译选项
编译选项

Rust 编译选项编译选项 RFC 编译选项编译选项编译选项 RFC 编译选项编译选项
编译选项编译选项编译选项编译选项编译选项编译选项编译选项 Rust 编译选项 RFC
编译选项编译选项编译选项

2.1.2

Rust 编译选项编译选项 Rust 编译选项 **rustc** 编译选项 Rust 编译选项编译选项
编译选项编译选项 .a .so .lib .dll 编译选项

rustc 编译选项

· rustc 编译选项编译选项 UNIX/Linux 编译选项 UNIX 编译选项 Windows
编译选项

· rustc 编译选项编译选项编译选项编译选项编译选项编译选项编译选项

· rustc 编译选项 LLVM 编译选项编译选项编译选项编译选项编译选项编译选项

· rustc 编译选项 Rust 编译选项编译选项 Rust 编译选项

· rustc 编译选项 Rust 编译选项编译选项编译选项编译选项编译选项 LLVM IR

· rustc 编译选项编译选项编译选项编译选项编译选项编译选项

2.1.3

Rust 编译选项编译选项编译选项编译选项 Rust 编译选项编译选项编译选项
Rust 编译选项编译选项编译选项编译选项编译选项编译选项编译选项 I/O

编译选项编译选项 [no_std] 编译选项编译选项编译选项编译选项编译选项编译选项
编译选项

· 编译选项 trait 编译选项 Copy 编译选项 Debug 编译选项 Display 编译选项 Option

· 编译选项编译选项 bool 编译选项 char 编译选项 i8/u8 编译选项 i16/u16 编译选项 i32/u32 编译选项 i64/u64
isize/usize 编译选项 f32/f64 编译选项 str 编译选项 array 编译选项 slice 编译选项 tuple 编译选项 pointer

- 標準ライブラリに `String` `Vec` `HashMap` `Rc` `Arc` `Box`
- `println` `assert` `panic` `vec`

2.1.4

- Rust
- `trait` API
- `I/O` `Sync` `trait` `TCP` `UDP` `I/O`
- `os`
- `std` `mem` `std` `ptr` `std` `intrinsics`
- `Option` `Result`

2.1.5

- `rs` `crate` Rust
- Rust `crates.io` `docs.rs`
- Rust `Cargo` Rust `Cargo` Ruby `bundler` Python `pip` Node.js `npm` `Cargo` Rust `Cargo` `Cargo` `2-1`
- `2-1` `cargo`

1. \$ cargo new bin_crate
2. \$ cargo new --lib lib_crate

2-1 **cargo new** **cargo build**
cargo new **--lib** **cargo build**
cargo run

2.2

Rust **Statement** **Expression**

Declaration statement **Expression statement**

· **Item**
extern **use**

·

2-2

2-2

1. // extern crate std;
2. // use std::prelude::v1::*;
3. fn main() {
4. pub fn answer() -> () {
5. let a = 40;
6. let b = 2;
7. assert_eq!(sum(a, b), 42);
8. }
9. pub fn sum(a: i32, b: i32) -> i32 {
10. a + b
11. }
12. answer();
13. }

2-2 1 2
prelude Rust crate
[no_std]

fn answer sum fn function

answer OCmal “”

answer let a b assert_eq
Rust Rust

sum i32 a b

2-2

Rust

Block Expression
answer
sum

Rust Rust

2.3

let Rust let
Binding Identifier Value

2.3.1

Rust 表达式分为 **Place Expression** 和 **Value Expression** 两种。LValue 和 RValue 分别

表示左值和右值。

- 常量
- 变量
- `expr*`
- `expr[expr]`
- `expr.field`
- 函数调用

表达式分为左值和右值。左值是指可以取地址的表达式，右值是指不能取地址的表达式。

左值

右值

常量表达式

常量表达式是指在编译时常量求值的表达式。常量表达式可以用于声明常量变量。

常量表达式函数是指函数体由常量表达式组成的函数。常量表达式函数可以用于声明常量变量。

常量表达式变量是指变量声明中的初始化表达式是常量表达式的变量。常量表达式变量可以用于声明常量变量。

- 常量表达式变量
- 常量表达式函数
- 常量表达式变量
- `match` 表达式

常量表达式变量是指变量声明中的初始化表达式是常量表达式的变量。常量表达式变量可以用于声明常量变量。

常量

常量表达式变量是指变量声明中的初始化表达式是常量表达式的变量。常量表达式变量可以用于声明常量变量。

```

1. pub fn temp() -> i32 {
2.     return 1;
3. }
4. fn main(){
5.     let x = &temp();
6.     temp() = *x; // error[E0070]: invalid left-hand side expression
7. }

```

Figure 2-3: `temp` in `main` is not a lvalue. `temp` is a function, not a variable, so it cannot be assigned to.

2.3.2 `let` and `mut`

Figure 2-4: `let` and `mut` are used to declare variables. `let` is used for immutable variables, and `mut` is used for mutable variables.

Figure 2-4: `let` and `mut` are used to declare variables

```

1. fn main(){
2.     let a = 1;
3.     // a = 2; // immutable and error
4.     let mut b = 2;
5.     b = 3; // mutable
6. }

```

Figure 2-4: `a` is an immutable variable, and `b` is a mutable variable. `mut` is used to declare a mutable variable.

Figure 2-4: `let` and `mut` are used to declare variables. `let` is used for immutable variables, and `mut` is used for mutable variables.

2.3.3 `const`

Figure 2-5: `const` is used to declare a constant variable. `const` is used to declare a variable that cannot be changed.

Figure 2-5: `const` is used to declare a constant variable

```

1. fn main(){
2.     let place1 = "hello";
3.     let place2 = "hello".to_string();
4.     let other = place1;
5.     println!("{:?}", other);
6.     let other = place2;
7.     println!("{:?}", other); // Err: other value used here after move
8. }

```

2-5 let place1 place2 place1
 other **place1** other place2 other place2
 other

5 other 7
 "other value used here after move"
 Rust
 3

Ownership Rust Move
 Copy Rust GC

Rust &
 2-6

2-6

```

1. fn main(){
2.     let a = [1,2,3];

```



```

3.     let b = &a;
4.     println!("{:p}", b); // 0x7ffc067704
5.     let mut c = vec![1,2,3];
6.     let d = &mut c;
7.     d.push(4);
8.     println!("{:?}", d); // [1, 2, 3, 4]
9.     let e = &42;
10.    assert_eq!(42, *e);
11. }

```

2-6 `a` のアドレスを `b` に格納し、`b` を使って `a` の値を出力する。このように、`println!` の第 2 引数に `&p` を指定すると、`p` が指している変数のアドレスを出力する。

また、`let mut c = &mut c` のように、変数 `c` のアドレスを `c` に格納し、`d.push(4)` のように、`d` を使って `c` の値を変更する。

42 のアドレスを `e` に格納し、`assert_eq!(42, *e)` のように、`e` が指している変数の値を確認する。

2-7 `main` 関数の内部で、`main` 関数のアドレスを `main` に格納する。

```

1. let mut _0: &i32;
2. let mut _1: i32;
3. _1 = const 42i32;
4. _0 = &_1;

```

2-7 `let e = &42` のように、`e` に `42` のアドレスを格納する。また、`*e` のように、`e` が指している変数の値を確認する。また、`&a` のように、`a` のアドレスを格納する。また、`&mut c` のように、`c` のアドレスを格納する。また、`a.c` のように、`a` の `c` 属性のアドレスを格納する。

2.4 関数

関数の定義は、`main` 関数のように、`main` 関数の内部で、`main` 関数のアドレスを `main` に格納する。

2.4.1 関数

fn 関数 Rust
fn 関数
関数

FizzBuzz FizzBuzz 3
fizz 5 buzz 3 5 fizzbuzz
2-8

2-8 FizzBuzz

```
1. pub fn fizz_buzz(num: i32) -> String {  
2.     if num % 15 == 0 {  
3.         return "fizzbuzz".to_string();  
4.     } else if num % 3 == 0 {  
5.         return "fizz".to_string();  
6.     } else if num % 5 == 0 {  
7.         return "buzz".to_string();  
8.     } else {  
9.         return num.to_string();  
10.    }  
11. }  
12. fn main(){  
13.     assert_eq!(fizz_buzz(15), "fizzbuzz".to_string());  
14.     assert_eq!(fizz_buzz(3), "fizz".to_string());  
15.     assert_eq!(fizz_buzz(5), "buzz".to_string());  
16.     assert_eq!(fizz_buzz(13), "13".to_string());  
17. }
```

2-8 fn fizz_buzz pub fn fizz_buzz
num i32 -> String i32 String
Rust

return 2-8

return return

2-8 `to_string` を使って `String` に変換する Rust のコードを例として、8 行のコードを実行する。

2.4.2 変数宣言

Rust の変数宣言の範囲を **Lexical Scope** と呼ぶ。変数宣言の範囲は、2-9 のコード

2-9 のコードを参照。

```
1. fn main(){
2.     let v = "hello world!";
3.     assert_eq!(v, "hello world!");
4.     let v = "hello Rust!";
5.     assert_eq!(v, "hello Rust!");
6.     {
7.         let v = "hello World!";
8.         assert_eq!(v, "hello World!");
9.     }
10.    assert_eq!(v, "hello Rust!");
11. }
```

2-9 のコードの 2 行と 5 行の `v` は `hello world` と `hello Rust` をそれぞれ宣言する。let で宣言した `v` は `hello Rust` の宣言で上書きされる。このように、**Variable Shadow** と呼ばれる現象が発生する。

6 行と 9 行の `let` は、`let` で宣言した `v` は `hello World` と宣言する。

10 行の `assert_eq` は `v` が `hello Rust` であることを確認する。

変数宣言の範囲を **LifeTime** と呼ぶ。変数宣言の範囲は、`let` で宣言した変数の範囲である。

2.4.3 変数

□□□□2-10□□□□□□□□□□□□□□□□

2-10

 $\frac{1}{\sqrt{2}}$

```
fn pointer
```

```

main math sum product sum
product fn i32 i32 - i32
math

```

2-11

2-11

```
1. fn is_true() -> bool { true }
2. fn true_maker() -> fn() -> bool { is_true }
3. fn main(){
4.     assert_eq!(true_maker()(), true);
5. }
```

2-11 `is_true` `true` `true_maker` `fn` `bool` `is_true`

`main` `true_maker` `true_maker` `true_maker` `is_true` `is_true` `true`

2.4.5 CTFE

Rust C++ D **Compile-Time Function Execution (CTFE)** Rust 2018 1.30 CTFE Rust 2018 Nightly Rust 2-12 CTFE `const fn`

2-12 `const fn`

```
1. //#[feature(const_fn)]
2. const fn init_len() -> usize {
3.     return 5;
4. }
5. fn main(){
6.     let arr = [0; init_len()];
7. }
```

2-12 `const fn` `init_len` 5 `main` `[0; N]` `0` `N` `N` `init_len`

Rust `init_len` CTFE Rust 2018 `[feature(const_fn)]` Rust 2015 `const fn` `fn` `const fn` `const`

`const fn` `const generics` `const generics` `impl T const N usize Foo for [T; N]{...}` `trait Foo`

Rust **CTFE** **miri** `miri` **MIR**
Rust `rustc` Rust `Rust`
Rust **CTFE**

2.4.6

-
-
-

2-13

2-13

```
1. fn main(){
2.     let out = 42;
3.     // fn add(i: i32, j: i32) -> i32 { i + j + out}
4.     fn add(i: i32, j: i32) -> i32 { i + j }
5.     let closure_annotated = |i: i32, j: i32| -> i32 { i + j + out};
6.     let closure_inferred = |i, j| i + j + out;
7.     let i = 1;
8.     let j = 2;
9.     assert_eq!(3, add(i, j));
10.    assert_eq!(45, closure_annotated(i, j));
11.    assert_eq!(45, closure_inferred(i, j));
12. }
```

2-13 `main` `add`
`closure_annotated``closure_inferred`

91
3`add``out`
56`out`

2-14

2-14

```

1. fn closure_math<F: Fn() -> i32>(op: F) -> i32 {
2.     op()
3. }
4. fn main() {
5.     let a = 2;
6.     let b = 3;
7.     assert_eq!(math(|| a + b), 5);
8.     assert_eq!(math(|| a * b), 6);
9. }

```

2-14 closure_math F Fn-i32 trait Fn-i32 trait

Rust trait main math ||a+b||a*b Fn-i32 math

2-15

2-15

```

1. fn two_times_impl() -> impl Fn(i32) -> i32 {
2.     let i = 2;
3.     move |j| j * i
4. }

5. fn main() {
6.     let result = two_times_impl();
7.     assert_eq!(result(2), 4);
8. }

```

2-15 impl Fn i32-i32 Fn i32-i32

two_times_impl move i Rust

move move
i
5

2.5

Rust Rust

2.5.1

if Rust
if
if 2-16
2-16 if

```
1. fn main() {  
2.     let n = 13;  
3.     let big_n = if (n < 10 && n > -10) {  
4.         10 * n  
5.     } else {  
6.         n / 2  
7.     };  
8.     assert_eq!(big_n, 6);  
9. }
```

2-16 big_n if n
n 13 Rust i32 if
n else n 2 6.5
if else

big_n Rust i32
n 2 Rust
big_n 6

2.5.2

Rust while loop for...in

for...in FizzBuzz 2-17

2-17 for...in FizzBuzz

```
1. fn main() {
2.     for n in 1..101 {
3.         if n % 15 == 0 {
4.             println!("fizzbuzz");
5.         } else if n % 3 == 0 {
6.             println!("fizz");
7.         } else if n % 5 == 0 {
8.             println!("buzz");
9.         } else {
10.            println!("{}", n);
11.        }
12.    }
13. }
```

2-17 for...in 1..101 Range
for for 5

while loop FizzBuzz
loop while true 2-18
while true

2-18 while true

```

1. fn while_true(x: i32) -> i32 {
2.     while true {
3.         return x+1;
4.     }
5. }
6. fn main(){
7.     let y = while_true(5);
8.     assert_eq!(y, 6);
9. }

```

2-18 while_true while true Rust

while true while_true i32 while true return i32

Rust while **Flow Sensitive** while while_true i32 CTFE while CTFE if true

2-19

2-19 while true

```

1. fn while_true(x: i32) -> i32 {
2.     while true {
3.         return x+1;
4.     }
5.     x
6. }

```

2-19 while_true 5 x i32 while true

2.5.3 match

Rust match 2-20

2-20 match

```
1. fn main() {
2.     let number = 42;
3.     match number {
4.         0 => println!("Origin"),
5.         1...3 => println!("All"),
6.         | 5 | 7 | 13 => println!("Bad Luck"),
7.         n @ 42 => println!("Answer is {}", n),
8.         _ => println!("Common"),
9.     }
10. }
```

2-20 match switch case

Rust match **Pattern Matching**

Rust match **Binding Mode** match

match let for

2.5.4 if let while let

Rust if let while let match if let

2-21 if let

```

1. fn main() {
2.     let boolean = true;
3.     let mut binary = 0;
4.     if let true = boolean {
5.         binary = 1;
6.     }
7.     assert_eq!(binary, 1);
8. }

```

2-21 if let match if let
 binary 0 boolean true binary 1

while let while let
 match 2-22

2-22 match

```

1. fn main() {
2.     let mut v = vec![1, 2, 3, 4, 5];
3.     loop {
4.         match v.pop() {
5.             Some(x) => println!("{}", x),
6.             None => break,
7.         }
8.     }
9. }

```

2-22 v pop
 loop v pop Option match Some
 x None Rust Option Some x
 None break

6 while let
 2-23

2-23 while let

```

1. fn main(){
2.     let mut v = vec![1,2,3,4,5];
3.     while let Some(x) = v.pop() {
4.         println!("{}", x);
5.     }
6. }

```

例2-23 while let 与 if let 结合使用。Some x 表示一个 Option 类型的值，其中 x 是一个值。println 函数用于打印输出。

2.6 布尔类型

Rust 中的布尔类型是 bool。

2.6.1 布尔类型

Rust 中的布尔类型是 bool，其值为 true 或 false。例2-24 展示了 bool 类型的用法。

例2-24 bool 类型

```

1. fn main(){
2.     let x = true;
3.     let y: bool = false;
4.     let x = 5;
5.     if x > 1 { println!( "x is bigger than 1")};
6.     assert_eq!(x as i32, 1);
7.     assert_eq!(y as i32, 0);
8. }

```

例2-24 展示了 bool 类型的用法。x 和 y 都是 bool 类型的变量。x 的值为 true，y 的值为 false。Rust 中的 bool 类型只有两个值：true 和 false。bool 类型可以与 i32 类型进行比较，因为 i32 类型的值 1 和 0 分别对应 true 和 false。Rust 中的 bool 类型可以与 i32 类型进行比较，因为 i32 类型的值 1 和 0 分别对应 true 和 false。

```

Rust
· Unsigned Integer Signed Integer
    >u8 0 2^8-1 1 u8 Rust I/O I/O u8
    >u16 0 2^16-1 2
    >u32 0 2^32-1 4
    >u64 0 2^64-1 8
    >u128 0 2^128-1 16
    >i8 -2^7 2^7-1 1
    >i16 -2^15 2^15-1 2
    >i32 -2^31 2^31-1 4
    >i64 -2^63 2^63-1 8
    >i128 -2^127 2^127-1 16
    ·
    >usize 0 2^32-1 0 2^64-1 4 8
    >isize -2^31 2^31-1 -2^63 2^63-1 4 8
    ·
    >f32 32 6 -3.4×10^38 3.4×10^38

```

>f64 64 15 -1.8×10³⁰⁸ 1.8×10³⁰⁸

2-25

2-25

```
1. fn main() {
2.     let num = 42u32;
3.     let num: u32 = 42;
4.     let num = 0x2A; // 十六进制
5.     let num = 0o106; // 八进制
6.     let num = 0b1101_1011; // 二进制
7.     assert_eq!(b'*', 42u8); // 字节字面量
8.     assert_eq!(b'\'', 39u8);
9.     let num = 3.1415926f64;
10.    assert_eq!(-3.14, -3.14f64);
11.    assert_eq!(2., 2.0f64);
12.    assert_eq!(2e4, 20000f64);
13.    println!("{:?}", std::f32::INFINITY);
14.    println!("{:?}", std::f32::NEG_INFINITY);
15.    println!("{:?}", std::f32::NAN);
16.    println!("{:?}", std::f32::MIN);
17.    println!("{:?}", std::f32::MAX);
18. }
```

2-25 42u32 u32 Rust i32

0x 0o 0b 0x2A 0o106 0b1101_1011

Rust b b * 42u8

Rust f64 std f32 std f64 IEEE INFINITY NEG_INFINITY NAN MIN MAX

2.6.3 字符串

Rust 字符串类型 Char 和 Unicode 字符串
4 字节 2-26 字节

2-26 字节

```
1. fn main(){
2.     let x = 'r';
3.     let x = 'Ú';
4.     println!("{}", '\');
5.     println!("{}", '\\');
6.     println!("{}", '\n');
7.     println!("{}", '\r');
8.     println!("{}", '\t');
9.     assert_eq!('\x2A', '*');
10.    assert_eq!('\x25', '%');
11.    assert_eq!('\u{CA0}', 'ø');
12.    assert_eq!('\u{151}', 'ö');
13.    assert_eq!('%' as i8, 37);
14.    assert_eq!('ø' as i8, -96);
15. }
```

2-26 字节 Unicode 字符串 Ú 字符串 * 字符串
Rust 字符串 4 字节 8 字节

ASCII 字符串 Unicode 字符串 2A ASCII 字符串 * 字符串
字符串 \xHH 字符串 151 Unicode 字符串 \u{HHH} 字符串
9 字符串 12 字符串

字符串 as 字符串 % 字符串 ASCII 字符串 37 字符串 i8
字符串 -96

2.6.4 数组

Array Rust 字符串

· 字符串


```

1. fn main(){
2.     assert_eq!((1..5), std::ops::Range{ start: 1, end: 5 });
3.     assert_eq!((1..=5), std::ops::RangeInclusive::new(1, 5));
4.     assert_eq!(3+4+5, (3..6).sum());
5.     assert_eq!(3+4+5+6, (3..=6).sum());
6.     for i in (1..5) {
7.         println!("{}", i); // 1,2,3,4
8.     }
9.     for i in (1..=5) {
10.        println!("{}", i); // 1,2,3,4,5
11.    }
12. }

```

2-28 `std::ops::Range` `std::ops::RangeInclusive` `sum` `for`

2.6.6 `Slice`

`Slice` `[T]` `&[T]` `&mut [T]`

2-29

2-29

```

1. fn main(){
2.     let arr: [i32; 5] = [1, 2, 3, 4, 5];

```

```

3.     assert_eq!(&arr, &[1, 2, 3, 4, 5]);
4.     assert_eq!(&arr[1..], [2, 3, 4, 5]);
5.     assert_eq!(&arr.len(), &5);
6.     assert_eq!(&arr.is_empty(), &false);
7.     let arr = &mut [1, 2, 3];
8.     arr[1] = 7;
9.     assert_eq!(arr, &[1, 7, 3]);
10.    let vec = vec![1, 2, 3];
11.    assert_eq!(&vec[..], [1, 2, 3]);
12. }

```

2-29 `&arr` `arr[1..]` `arr` `1`
`const fn len is_empty`
`&mut` `7` `9`
`vec` `10` `11`

2.6.7 str

Rust `str` `&str` **Rust** `String` `str` `2-30`

2-30 `str`

Rust `*const T` `*mut T`

2-31

2-31

```
1. fn main() {
2.     let mut x = 10;
3.     let ptr_x = &mut x as *mut i32;
4.     let y = Box::new(20);
5.     let ptr_y = &*y as *const i32;
6.     unsafe {
7.         *ptr_x += *ptr_y;
8.     }
9.     assert_eq!(x, 30);
10. }
```

2-31 `as` `&mut x` `*mut i32` `ptr_x` 2 3

4 `Box::new(20)` 20 `ptr_y`

`ptr_x` `ptr_y` 30 6 8 `unsafe`

13

2.6.9 never

Rust `never` Rust

never 2-32

2-32 `neüer`

```

1.  #![feature(never_type)]
2.  fn foo() -> u32 {
3.      let x: ! = {
4.          return 123
5.      };
6.  }
7.  fn main() {
8.      let num: Option<u32> = Some(42);
9.      match num {
10.         Some(num) => num,
11.         None => panic!("Nothing!"),
12.     };
13. }

```

2-32 [feature never_type] never
 Nightly never

2 6 foo x never
 return return 123 x never
 return break continue

main match None panic
 match panic never
 Some num u32 never

2.7

Rust 4

- Tuple
- Struct
- Enum
- Union

Rust

- Named-Field Struct
- Tuple-Like Struct
- Unit-Like Struct

2-34

2-34

```
1. #[derive(Debug, PartialEq)]
2. struct People {
3.     name: &'static str,
4.     gender: u32,
5. }
6. impl People {
7.     fn new(name: &'static str, gender: u32) -> Self{
8.         return People{name: name, gender: gender};
9.     }
10.    fn name(&self) {
11.        println!("name: {:?}", self.name);
12.    }
13.    fn set_name(&mut self, name: &'static str) {
14.        self.name = name;
15.    }
16.    fn gender(&self){
17.        let gender = if (self.gender == 1) {"boy"} else {"girl"};
18.        println!("gender: {:?}", gender);
19.    }
20. }
```

2-34 struct People should have a camel case name

name type name type

例2-36

```
1. struct Color(i32, i32, i32);
2. fn main(){
3.     let color = Color(0, 1, 2);
4.     assert_eq!(color.0, 0);
5.     assert_eq!(color.1, 1);
6.     assert_eq!(color.2, 2);
7. }
```

例2-36は、`Color`構造体を宣言し、`main`関数内でそのインスタンスを作成し、各フィールドの値が期待通りであることを確認しています。

例2-37は、`New Type`を宣言し、`main`関数内でそのインスタンスを作成し、各フィールドの値が期待通りであることを確認しています。

例2-37 New Type

```
1. struct Integer(u32);
2. type Int = i32;
3. fn main(){
4.     let int = Integer(10);
5.     assert_eq!(int.0, 10);
6.     let int: Int = 10;
7.     assert_eq!(int, 10);
8. }
```

例2-37は、`Integer`構造体を宣言し、`u32`型をフィールドとして宣言しています。また、`New Type`として`Int`を宣言し、`i32`型と等価であることを示しています。

例2-37では、`type`キーワードを使用して、`Int`という新しい型を宣言しています。この型は、`i32`型と等価であることを示しています。また、`Integer`構造体のフィールド`0`は、`i32`型であることを示しています。

Rustでは、`Int`という型を宣言し、`i32`型と等価であることを示しています。

例2-38

```

1. struct Empty;
2. fn main() {
3.     let x = Empty;
4.     println!("{:p}", &x);
5.     let y = x;
6.     println!("{:p}", &y);
7.     let z = Empty;
8.     println!("{:p}", &z);
9.     assert_eq!(.., std::ops::RangeFull);
10. }

```

图2-38展示了Empty结构体的定义。struct Empty{}定义了一个空结构体。在main函数中，我们创建了一个Empty类型的变量x，并打印了其地址。然后，我们创建了一个指向x的变量y，并打印了其地址。最后，我们创建了一个Empty类型的变量z，并打印了其地址。由于x和y都指向同一块内存，因此它们的地址是相同的。而z是一个新的Empty类型的变量，因此它的地址是不同的。最后，我们使用assert_eq!断言了..和std::ops::RangeFull是相等的。

图5展示了x和y都指向同一块内存的情况。x和y都是Empty类型的变量，它们的地址是相同的。而z是一个新的Empty类型的变量，因此它的地址是不同的。

图7展示了z的地址。z是一个新的Empty类型的变量，因此它的地址是不同的。我们使用println!打印了x、y和z的地址。x和y的地址是相同的，而z的地址是不同的。

- Debug模式下x、y、z的地址
- Release模式下x、y、z的地址

图展示了Release和Debug模式下的地址。在Release模式下，x和y的地址是相同的，而z的地址是不同的。在Debug模式下，x和y的地址是相同的，而z的地址是不同的。

图展示了New Type类型的定义。New Type类型是一种特殊的类型，它允许我们定义一个类型，该类型与另一个类型具有相同的内存布局，但具有不同的语义。在图中，我们定义了一个New Type类型的变量，并打印了其地址。然后，我们使用RangeFull断言了..和std::ops::RangeFull是相等的。

2.7.3 枚举

图展示了Enum类型的定义。Enum类型是一种特殊的类型，它允许我们定义一个类型，该类型具有有限的、离散的值。在图中，我们定义了一个Enum类型的变量，并打印了其地址。然后，我们使用enum断言了..和std::ops::RangeFull是相等的。

图2-39展示了Enum类型的定义。

```

1.  enum Number {
2.      Zero,
3.      One,
4.      Two,
5.  }
6.  fn main() {
7.      let a = Number::One;
8.      match a {
9.          Number::Zero => println!("0"),
10.         Number::One  => println!("1"),
11.         Number::Two  => println!("2"),
12.     }
13. }

```

2-39 Rust 枚举类型 Number 枚举类型 Zero One Two

main 函数中匹配 Number 枚举类型的 7 个分支

Rust 枚举类型 C 语言枚举类型

2-40 Rust 枚举类型 C 语言枚举类型

```

1.  enum Color {
2.      Red = 0xff0000,
3.      Green = 0x00ff00,
4.      Blue = 0x0000ff,
5.  }
6.  fn main() {
7.      println!("roses are #{:06x}", Color::Red as i32);
8.      println!("violets are #{:06x}", Color::Blue as i32);
9.  }

```

2-40 Rust 枚举类型 Color 枚举类型 Red Green Blue

Rust 枚举类型 C 语言枚举类型 2-41

例2-41

```
1.  enum IpAddr {
2.      V4(u8, u8, u8, u8),
3.      V6(String),
4.  }
5.  fn main(){
6.      let x : fn(u8, u8, u8, u8) -> IpAddr = IpAddr::V4;
7.      let y : fn(String) -> IpAddr = IpAddr::V6;
8.      let home = IpAddr::V4(127, 0, 0, 1);
9.  }
```

例2-41 `IpAddr` の定義と、`main` の実行結果

6行目の `V4` は `fn` の型で、`u8` の型を4つ指定して `IpAddr` の型に型変換する。7行目の `V6` は `fn` の型で、`String` の型を指定して `IpAddr` の型に型変換する。

8行目の `home` は `IpAddr` の型で、`V4` の型変換関数を使って、`127, 0, 0, 1` の4つの `u8` の型に変換する。

Rust の `enum` は、`enum` の型変換関数を使って、`enum` の型に変換する。Rust の `enum` は、`enum` の型変換関数を使って、`enum` の型に変換する。

例2-42

```
1.  enum Option{
2.      Some(i32),
3.      None,
4.  }
5.  fn main(){
6.      let s = Some(42);
7.      let num = s.unwrap();
8.      match s {
9.          Some(n) => println!("num is: {}", n),
10.         None => (),
11.     };
12. }
```

例2-42 `Option` の定義と、`main` の実行結果

`Some(i32)` の型変換関数を使って、`Some(i32)` の型に変換する。

Option 类型，它包含一个值，或者没有值。它有两种状态：Some 和 None。Some 表示有值，None 表示没有值。

在 main 函数中，我们定义了一个变量 s，它的类型是 Option<i32>。我们使用 unwrap 方法来获取 s 的值。如果 s 是 Some(42)，那么 unwrap 会返回 42。如果 s 是 None，那么 unwrap 会 panic。

Option 类型是 Rust 中的一个非常常用的类型。它可以帮助我们处理可能为 null 的值。在 Rust 2.43 版本之前，我们使用 Option 类型来处理可能为 null 的值。在 Rust 2.43 版本之后，我们使用 Option 类型来处理可能为 null 的值。

2-43 Option<T>

```
1. fn main() {
2.     let s: &Option<String> = &Some("hello".to_string());
3.     // Rust 2015 版本
4.     match s {
5.         &Some(ref s) => println!("s is: {}", s),
6.         _ => (),
7.     };
8.     // Rust 2018 版本
9.     match s {
10.        Some(s) => println!("s is: {}", s),
11.        _ => (),
12.    };
13. }
```

在 Rust 2.43 版本之前，我们使用 Option 类型来处理可能为 null 的值。在 Rust 2.43 版本之后，我们使用 Option 类型来处理可能为 null 的值。

在 Rust 2015 版本之前，我们使用 Option 类型来处理可能为 null 的值。在 Rust 2015 版本之后，我们使用 Option 类型来处理可能为 null 的值。

在 Rust 2018 版本之前，我们使用 Option 类型来处理可能为 null 的值。在 Rust 2018 版本之后，我们使用 Option 类型来处理可能为 null 的值。

2.8 集合

Rustのstd collectionsには4つのコレクションがある

- ・ Vec VecDeque LinkedList
- ・ Key-Value HashMap BTreeMap
- ・ HashSet BTreeSet
- ・ BinaryHeap

2.8.1 Vec

2-44のVecは、動的に成長可能な配列である。2-44のVecは、動的に成長可能な配列である。

2-44のVecは、動的に成長可能な配列である。

```
1. fn main() {
2.     let mut v1 = vec![];
3.     v1.push(1);
4.     v1.push(2);
5.     v1.push(3);
6.     assert_eq!(v1, [1, 2, 3]);
7.     assert_eq!(v1[1], 2);
8.     let mut v2 = vec![0; 10];
9.     let mut v3 = Vec::new();
10.    v3.push(4);
11.    v3.push(5);
12.    v3.push(6);
13.    // v3[4]; error: index out of bounds
14. }
```

2-44のVecは、動的に成長可能な配列である。v1, v2, v3は、動的に成長可能な配列である。mutは、可変参照を示す。Vec::new()は、空のVecを作成する。

vecは、動的に成長可能な配列である。pushは、Vecに要素を追加する。8は、Vecの容量を示す。

□□□□

Rust□□□□□□□□□□□□□□□□□□□□□□□□13□□□□□□v3[4]□□□□□□
panic□□□thread□main□panicked at□index out of bounds□

2.8.2 □□□□□□□□

□□□□□Double-ended Queue□□□□Deque□□□□□□□□□□□□□□□□
□□□
□□

Rust□□VecDeque□□□□□□□RingBuffer□□□□□□□□□□□□□□□□2-
45□□□□□□□□□□□□□□

□□□□2-45□VecDeque□T□□□

```
1. use std::collections::VecDeque;
2. fn main () {
3.     let mut buf = VecDeque::new();
4.     buf.push_front(1);
5.     buf.push_front(2);
6.     assert_eq!(buf.get(0), Some(&2));
7.     assert_eq!(buf.get(1), Some(&1));
8.     buf.push_back(3);
9.     buf.push_back(4);
10.    buf.push_back(5);
11.    assert_eq!(buf.get(2), Some(&3));
12.    assert_eq!(buf.get(3), Some(&4));
13.    assert_eq!(buf.get(4), Some(&5));
14. }
```

□□□□□ 2-45 □□□□□□□ use □□□□□□ std□□collections□□
VecDeque□□□VecDeque□T□□□□□□Vec□T□□□□□□□□□□□□

□□□□VecDeque□□□□□□push□□□□push_front□push_back□
push_front□□□□□□□push_back□□□□□□□□□□get□□□□□□□□□□□□□□□□□□
□□□

□□□6□□□7□□□□push_front□□□□□□□□1□2□□□□□□□□□□1□0□□□□□□
□□□□□□□□□□□□

1113push_back345234

2.8.3

RustVecVecDequeCPU

2-46

2-46LinkedListT

```
1. use std::collections::LinkedList;;
2. fn main() {
3.     let mut list1 = LinkedList::new();
4.     list1.push_back('a');
5.     let mut list2 = LinkedList::new();
6.     list2.push_back('b');
7.     list2.push_back('c');
8.     list1.append(&mut list2);
9.     println!("{:?}", list1); // ['a', 'b', 'c']
10.    println!("{:?}", list2); // []
11.    list1.pop_front();
12.    println!("{:?}", list1); // ['b', 'c']
13.    list1.push_front('e');
14.    println!("{:?}", list1); // ['e', 'b', 'c']
15.    list2.push_front('f');
16.    println!("{:?}", list2); // ['f']
17. }
```

2-46usestdcollectionsLinkedListpush_backpush_frontappend

2.8.4 Key-ValueHashMapBTreeMap

RustKey-Value

- **HashMap** **K**, **V**
- **BTreeMap** **K**, **V**

Key **Value**
HashMap **BTreeMap** **HashMap** **K**
V **BTreeMap** **K** **V** 2-47

2-47 **HashMap** **K** **V** **BTreeMap** **K** **V**

```
1. use std::collections::BTreeMap;
2. use std::collections::HashMap;
3. fn main() {
4.     let mut hmap = HashMap::new();
5.     let mut bmap = BTreeMap::new();
6.     hmap.insert(3, "c");
7.     hmap.insert(1, "a");
8.     hmap.insert(2, "b");
9.     hmap.insert(5, "e");
10.    hmap.insert(4, "d");
11.    bmap.insert(3, "c");
12.    bmap.insert(2, "b");
13.    bmap.insert(1, "a");
14.    bmap.insert(5, "e");
15.    bmap.insert(4, "d");
16.    println!("{:?}", hmap);
17.    println!("{:?}", bmap);
18. }
```

2-47 use std::collections::BTreeMap
use std::collections::HashMap
new
hmap bmap insert

16 hmap {1 a 2 b 3 c 5 e 4 d
} key HashMap

17 bmap {1 a 2 b 3 c 4 d 5 e
} BTreeMap

8

2.8.5 HashSetとBTreeSet

HashSet<K> と BTreeSet<K> はどちらも HashMap<K, V> と BTreeMap<K, V>のValue部分に置き換わればHashSet<K>とBTreeSet<K>になる。

- ・ どちらもKey-ValueのKey部分
- ・ どちらもValue部分
- ・ HashSetとBTreeSetの違い

HashSet<K>とBTreeSet<K>のサイズは2-48

2-48HashSet<K>BTreeSet<K>

```
1. use std::collections::HashSet;
2. use std::collections::BTreeSet;
3. fn main() {
4.     let mut hbooks = HashSet::new();

5.     let mut bbooks = BTreeSet::new();
6.     hbooks.insert("A Song of Ice and Fire");
7.     hbooks.insert("The Emerald City");
8.     hbooks.insert("The Odyssey");
9.     if !hbooks.contains("The Emerald City") {
10.         println!("We have {} books, but The Emerald City ain't one.",
11.             hbooks.len()
12.         );
13.     }
14.     println!("{:?}", hbooks);
15.     bbooks.insert("A Song of Ice and Fire");
16.     bbooks.insert("The Emerald City");
17.     bbooks.insert("The Odyssey");
18.     println!("{:?}", bbooks);
19. }
```

2-4814hbooksHashSet

18 books { A Song of Ice and Fire The Emerald City The Odyssey } BTreeSet

2.8.6 BinaryHeap

Rust BinaryHeap 2-49

2-49 BinaryHeap T

```
1. use std::collections::BinaryHeap;
2. fn main() {
3.     let mut heap = BinaryHeap::new();
4.     assert_eq!(heap.peek(), None);
5.     let arr = [93, 80, 48, 53, 72, 30, 18, 36, 15, 35, 45];
6.     for &i in arr.iter() {
7.         heap.push(i);
8.     }
9.     assert_eq!(heap.peek(), Some(&93));
10.    // [93, 80, 48, 53, 72, 30, 18, 36, 15, 35, 45]
11.    println!("{:?}", heap);
12. }
```

2-49 BinaryHeap new peek

4 peek None

5 8 push peek

93

BinaryHeap

2.9

Smart Pointer Rust C++ Rust Rust

Rust

Rust Box T Box T

Box T T
Box T
Rust Box T Box T
2-50

2-50 Box T

```
1. fn main() {
2.     #[derive(PartialEq)]
3.     struct Point {
4.         x: f64,
5.         y: f64,
6.     }
7.     let boxed_point = Box::new(Point { x: 0.0, y: 0.0 });
8.     let unboxed_point: Point = *boxed_point;
9.     assert_eq!(unboxed_point, Point { x: 0.0, y: 0.0 });
10. }
```

2-50 main Point Box new
Point
Box T
5

2.10 trait

trait Rust

Rust
Rust
Rust

trait Rust Haskell Typeclass 1
trait Rust

- trait Rust
-
- “”

trait

2.10.1

Rust Option T Vec T HashMap K V Box T Option T 2-51

2-51 Option T

```
1. // std::option::Option
2. enum Option<T>{
3.     Some(T),
4.     None,
5. }
```

T Option T Option u32 Option String Option T use std prelude v1 Rust Some T None Option T Option Some T Option None

Option T 2-52

2-52 Option T

```

1. use std::fmt::Debug;
2. fn match_option<T: Debug>(o: Option<T>) {
3.     match o {
4.         Some(i) => println!("{:?}", i),
5.         None => println!("nothing"),
6.     }
7. }
8. fn main() {
9.     let a: Option<i32> = Some(3);
10.    let b: Option<&str> = Some("hello");
11.    let c: Option<char> = Some('A');
12.    let d: Option<u32> = None;
13.    match_option(a); // 3
14.    match_option(b); // "hello"
15.    match_option(c); // 'A'
16.    match_option(d); // nothing
17. }

```

2-52 `match_option` trait `T: Debug` trait `Debug` trait `{}`

`Debug` `T` cannot be formatted using `Rust`

9 12 `a b c d` 4 `Option` `T` 4 `Rust` 4 `Option` `i32` `Option` `&str` `Option` `char` `Option` `u32` 4

2.10.2 trait

trait trait 2-53

2-53 **trait**

```

1. struct Duck;
2. struct Pig;
3. trait Fly {
4.     fn fly(&self) -> bool;
5. }
6. impl Fly for Duck {
7.     fn fly(&self) -> bool {
8.         return true;
9.     }
10. }
11. impl Fly for Pig {
12.     fn fly(&self) -> bool {
13.         return false;
14.     }
15. }
16. fn fly_static<T: Fly>(s: T) -> bool {
17.     s.fly()
18. }
19. fn fly_dyn(s: &Fly) -> bool {
20.     s.fly()
21. }
22. fn main() {
23.     let pig = Pig;
24.     assert_eq!(fly_static::<Pig>(pig), false);
25.     let duck = Duck;
26.     assert_eq!(fly_static::<Duck>(duck), true);
27.     assert_eq!(fly_dyn(&Pig), false);
28.     assert_eq!(fly_dyn(&Duck), true);
29. }

```

2-53 1 2 Duck Pig

3 5 trait Fly trait **Rust** trait


```
fn fly(&mut self) {
    // ...
}

trait Fly {
    fn fly(&mut self);
}
```

```
impl Fly for Duck {
    fn fly(&mut self) {
        println!("Duck fly");
    }
}
```

```
impl Fly for Pig {
    fn fly(&mut self) {
        println!("Pig fly");
    }
}
```

```
fn main() {
    let mut duck = Duck { .. };
    duck.fly();
}
```

```
fn fly_static(&mut self) {
    // ...
}

impl Fly for T {
    fn fly(&mut self) {
        fly_static(self);
    }
}
```

```
fn fly_dyn(&mut self) {
    // ...
}
```

```
fn main() {
    fly_static(&mut duck);
    fly_dyn(&mut duck);
}
```

```
let mut pig = Pig { .. };
assert!(fly_static(&mut pig));
assert!(fly_dyn(&mut pig));
```

```
fly_static(&mut duck);
fly_dyn(&mut duck);
```

Rust 中 `fly_static` 和 `fly_dyn` 的区别在于前者是静态方法，后者是动态方法。静态方法在编译时就被确定，而动态方法在运行时才被确定。

```
fly_dyn(&mut pig);
fly_dyn(&mut duck);
```

Rust trait C++ trait
 Rust C/C++ trait

Rust trait trait
 Debug trait println {}
 2-54

2-54 Debug trait

```
1. use std::fmt::*;
2. struct Point {
3.     x: i32,
4.     y: i32,
5. }
6. impl Debug for Point {
7.     fn fmt(&self, f: &mut Formatter) -> Result {
8.         write!(f, "Point {{ x: {}, y: {} }}", self.x, self.y)
9.     }
10. }
11. fn main(){
12.     let origin = Point { x: 0, y: 0 };
13.     println!("The origin is: {:?}", origin);
14. }
```

2-54 Point Point Debug trait
 use std::fmt::Debug

Debug trait fmt Point 6 10
 main println Point origin

[derive Debug] Debug trait
 Rust 12
 3 trait

2.11

Rust Result TE Result TE
 Option T

2-55 Result T E

2-55 Result T E

```
1.  enum Result<T, E> {
2.      Ok(T),
3.      Err(E),
4.  }
```

Option T Result T E E
Error Result T E 2-56

2-56 Result T E

```
1.  fn main(){
2.      let x: Result<i32, &str> = Ok(-3);
3.      assert_eq!(x.is_ok(), true);
4.      let x: Result<i32, &str> = Err("Some error message");
5.      assert_eq!(x.is_ok(), false);
6.  }
```

2-56 Ok -3 Err Some error message
is_ok Ok T

Option T Result T E
Result T E
Option T

Rust 2015 main Result T E
Rust 2018 main Result T E 2-57

2-57 main Result T E

```
1.  // Rust 2018 版本
2.  use std::fs::File;
3.  fn main() -> Result<(), std::io::Error> {
4.      let f = File::open("bar.txt")?;
5.      Ok(())
6.  }
```

Figure 2-57: `main` calls `File::open` to open the file `test.txt`. The `std::io::Error` type is used to represent the error.

Figure 2-58: The `main` function returns 9.

2.12 Error Handling

The `Rust` compiler uses the `Result` type to represent the result of a function. The `Result` type is defined in the `std::result` module.

Figure 2-1: The `Result` type.

> `/*...*/` 多行注释

· 使用 `Markdown` 生成 `rustdoc` 的 `HTML`

> `///` 单行注释

> `//` 单行注释

`2-58`

`2-58`

```
1.  /// # 文档注释: Sum 函数
2.  /// 该函数为求和函数
3.  /// # usage:
4.  ///     assert_eq!(3, sum(1, 2));
5.  fn sum(a: i32, b: i32) -> i32 {
6.      a + b
7.  }
8.  fn main() {
9.      // 这是单行注释的示例
10.     /*
11.      * 这是区块注释, 被包含的区域都会被注释
12.      * 你可以把/* 区块 */ 置于代码中的任何位置
13.      */
14.     /*
15.      注意上面区块注释中的*符号纯粹是一种注释风格,
16.      实际并不需要
17.      */
18.     let x = 5 + /* 90 + */ 5;
19.     assert_eq!(x, 10);
20.     println!("2 + 3 = {}", sum(2, 3));
21. }
```

`2-58` `cargo doc` 生成 `HTML`

`Rust` 9

println
println

- `nothing` `Display` `println` `{ }` `2`
- `Debug` `println` `{ }` `2`
- `o` `println` `{o}` `2`
- `x` `println` `{x}` `2`
- `X` `println` `{X}` `2`
- `p` `println` `{p}` `2`
- `b` `println` `{b}` `2`
- `e` `println` `{e}` `2`
- `E` `println` `{E}` `2`

2.14

Rust Rust Rust

Rust Rust Rust CTFE Rust CTFE

if Rust Rust Option if let while let Rust while CTFE loop

Rust Rust Rust Rust

3

```

01
01
20 50 FORTRAN

```

3.1 实验目的

```
Rust [REDACTED] u32 [REDACTED] Rust [REDACTED] 4 [REDACTED]
```

[illegible][illegible]

3.1.1 背景

[illegible][illegible]

Ruby Python 的鸭子类型 Duck Typing 在 Ruby Python 中，只要一个对象实现了你所需要的方法，那么它就可以被当作该类型使用。

在 Ruby 中，你可以使用 `is_a?` 方法来检查一个对象是否是某个类的实例。而在 Python 中，你可以使用 `isinstance` 方法。

3.1.3 多态性

多态性是指一个对象可以具有多种不同的形态。在编程中，这通常意味着一个变量可以指向不同类型的对象，或者一个方法可以接受不同类型的参数。

多态性可以分为几种类型：
- **Parametric polymorphism**：参数多态性，通常通过泛型实现。
- **Ad-hoc polymorphism**：特设多态性，通过重载实现。
- **Subtype polymorphism**：子类型多态性，通过继承实现。
- **Static Polymorphism**：静态多态性，编译时确定的多态性。
- **Dynamic Polymorphism**：动态多态性，运行时确定的多态性。
Ad-hoc 多态性在 Rust 中通过 trait 实现，而在 Haskell 中通过 Typeclass 实现。

在 Rust 中，trait 是一种抽象的接口，它定义了一组方法和属性。实现 trait 的类或结构体必须实现 trait 中定义的所有方法。

Ad-hoc 多态性在 Rust 中通过 trait 实现，而在 Haskell 中通过 Typeclass 实现。在 Rust 中，trait 是一种抽象的接口，它定义了一组方法和属性。实现 trait 的类或结构体必须实现 trait 中定义的所有方法。

在 Rust 中，trait 是一种抽象的接口，它定义了一组方法和属性。实现 trait 的类或结构体必须实现 trait 中定义的所有方法。在 Rust 中，trait 是一种抽象的接口，它定义了一组方法和属性。实现 trait 的类或结构体必须实现 trait 中定义的所有方法。

3.2 Rust 类型系统

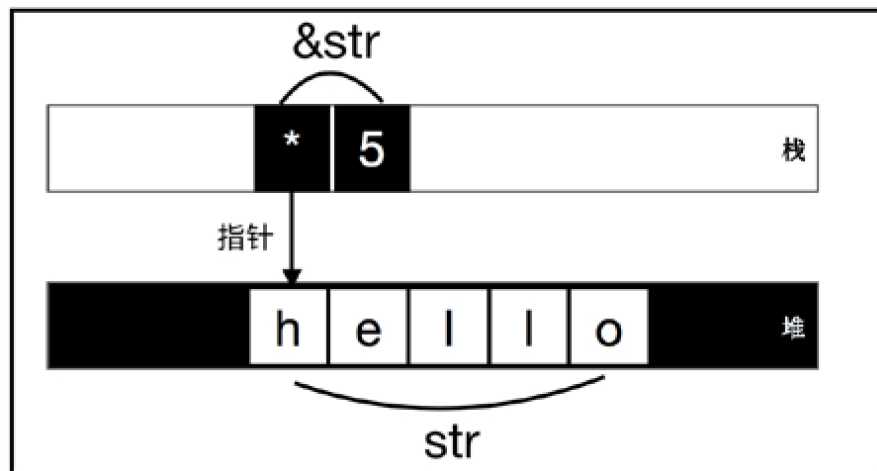
Rust 的类型系统是静态的，这意味着类型是在编译时确定的。Rust 的类型系统非常强大，它支持多种类型，包括基本类型、复合类型、函数类型等。

Rust 4
 Rust Option T Result T E
 break
 continue
 never Rust
 Rust

3.2.1

Rust Rust
 GC Rust LLVM IR
 3

Rust **Sized Type**
 u32 4 u64 8 Rust
Dynamic Sized Type DST str
 str
 Rust &str
 3-1



3-1 &str

&str str 4
 &str 3-1 str

このコードは、文字列 `str` のアドレスと長さを取得し、`&str` と `str.len()` を出力しています。

例3-1 `&str` の取得

```
1. fn main() {
2.     let str = "Hello Rust";
3.     let ptr = str.as_ptr();
4.     let len = str.len();
5.     println!("{:p}", ptr); // 0x555db4b96c00
6.     println!("{:?}", len); // 10
7. }
```

例3-1では、文字列 `str` のアドレスを `as_ptr()` で取得し、`len` は文字列の長さです。このように、文字列のアドレスと長さを取得する型は **Fat Pointer** と呼ばれます。

このように、`Rust` の `[T]` は、配列の型を表します。例3-2では、配列の初期化とリセットを行います。

例3-2 配列の初期化とリセット

```
1. fn reset(mut arr: [u32]) {
2.     arr[0] = 5;
3.     arr[1] = 4;
4.     arr[2] = 3;
5.     arr[3] = 2;
6.     arr[4] = 1;
7.     println!("reset arr {:?}", arr);
8. }
9. fn main() {
10.    let arr: [u32] = [1, 2, 3, 4, 5];
11.    reset(arr);
12.    println!("origin arr {:?}", arr);
13. }
```

例3-2 配列の初期化

```
fn reset(mut arr: [u32]) {  
    |          ^^^^^^^ `[u32]` does not have a constant size known at  
compile-time
```

3-3
[u32; 5] 3-3

3-3 [u32; 5]

```
1. fn reset(mut arr: [u32; 5]) {  
2.     arr[0] = 5;  
3.     arr[1] = 4;  
4.     arr[2] = 3;  
5.     arr[3] = 2;  
6.     arr[4] = 1;  
7.     println!("reset arr {:?}", arr); // [5, 4, 3, 2, 1]  
8. }  
9. fn main() {  
10.    let arr: [u32; 5] = [1, 2, 3, 4, 5];  
11.    reset(arr);  
12.    println!("origin arr {:?}", arr); // [1, 2, 3, 4, 5]  
13. }
```

3-3
Copy trait
[u32] [u32; 5]

3-2 &str
&mut [u32] &mut [u32] [u32] &[u32]
3-4

3-4 &mut [u32]

```

1.  fn reset(arr: &mut [u32]) {
2.      arr[0] = 5;
3.      arr[1] = 4;
4.      arr[2] = 3;
5.      arr[3] = 2;
6.      arr[4] = 1;
7.      // 重置之后, 原始数组为 [5, 4, 3, 2, 1]
8.      println!("array length {:?}", arr.len());
9.      // arr 已被重置为 [5, 4, 3, 2, 1]
10.     println!("reset array {:?}", arr);
11. }

12. fn main() {
13.     let mut arr = [1, 2, 3, 4, 5];
14.     // 重置之前, 原始数组为 [1, 2, 3, 4, 5]
15.     println!("reset before : origin array {:?}", arr);
16.     {
17.         let mut_arr: &mut[u32] = &mut arr;
18.         reset(mut_arr);
19.     }
20.     println!("reset after : origin array {:?}", arr);
21. }

```

3-4 `&mut [u32]` `&[u32]`
`&mut [u32]` `reset`

3-5 `&[u32[5]]` `&mut [u32]`

3-5 `&[u32[5]]` `&mut [u32]`

```

1.  fn main() {
2.      assert_eq!(std::mem::size_of::<[u32; 5]>(), 8);
3.      assert_eq!(std::mem::size_of::<&mut [u32]>(), 16);
4.  }

```

3-5 `std::mem::size_of` `&[u32[5]]`
`8` `16` `&[u32[5]]` `8` `&mut [u32]`

16

DST Rust Zero Sized Type
ZST 3-6

3-6

```
1. enum Void {}
2. struct Foo;
3. struct Baz {
4.     foo: Foo,
5.     qux: (),
6.     baz: [u8; 0],
7. }
8. fn main() {
9.     assert_eq!(std::mem::size_of::<()>(), 0);
10.    assert_eq!(std::mem::size_of::<Foo>(), 0);
11.    assert_eq!(std::mem::size_of::<Baz>(), 0);
12.    assert_eq!(std::mem::size_of::<Void>(), 0);
13.    assert_eq!(std::mem::size_of::<[(); 10]>(), 0);
14. }
```

3-6
ZST ZST
“”

3-7

3-7

```
1. fn main() {
2.     let v: () = vec![(); 10];
3. }
```

代码清单 3-7 编译会报错，如下：

```
|     let v: () = vec![(); 10];
|                               ^^^^^^^^^^^^^ expected (), found struct `std::vec::Vec`
```

std::vec::Vec<T> vec!(10);

3-8 Vec

3-8 Vec

```
1. fn main() {
2.     let v: Vec<()> = vec![(10)];
3.     for i in v {
4.         println!("{}", i);
5.     }
6. }
```

3-8 Vec 10

2 Rust HashSet T BTreeSet T HashMap K T HashMap K T HashMap K T HashSet T

Bottom Type 2 never

-
-

ZST “” “”

Rust Bang Type Rust

- **Diverging Function**
- continue break
- loop
- enum Void{ }

panic! This function never returns std::process::exit

`continue` `break` `loop`

Rust `if` `3-9`

`3-9`

```
1.  #![feature(never_type)]
2.  fn foo() -> ! {
3.      // ...
4.      loop { println!("jh"); }
5.  }
6.  fn main() {
7.      let i = if false {
8.          foo();
9.      } else {
10.         100
11.     };
12.     assert_eq!(i, 100);
13. }
```

`3-9` `if` `foo` `else`

`enum Void{}` `3-10`

`3-10`

```
1.  enum Void {}
2.  fn main() {
3.      let res: Result<u32, Void> = Ok(0);
4.      let Ok(num) = res;
5.  }
```

Rust `Result` `Ok` `Err` `enum Void{}` `Err` `if let`


```

1. fn main() {
2.     let x = "1";
3.     println!("{}", x.parse().unwrap());
4. }

```

3-12

```

error[E0284]: type annotations required
  |         println!("{}", x.parse().unwrap());
  |                               ^^^^^

```

3-12 Rust 1 parse u32 i32 Rust 3-13

3-13

```

1. fn main() {
2.     let x = "1";
3.     let int_x: i32 = x.parse().unwrap();
4.     assert_eq!(int_x, 1);
5. }

```

Rust 3-14

3-14

```

1. fn main() {
2.     let x = "1";
3.     assert_eq!(x.parse::<i32>().unwrap(), 1);
4. }

```

3-14 parse i32 3-13 3 turbofish

Rust 3-15

3-15

```

1. fn main() {
2.     let a = 0;
3.     let a_pos = a.is_positive();
4. }

```

3-15 `is_positive` Rust

error[E0599]: no method named `is_positive` found for type `{integer}` in the current scope

`{integer}` `a` Rust

3.3

Generic `Box` `T` `Option` `Result` `E`

3.3.1

3-16

```

1. fn foo<T>(x: T) -> T {
2.     return x;
3. }
4. fn main() {
5.     assert_eq!(foo(1), 1);
6.     assert_eq!(foo("hello"), "hello");
7. }

```

3-17

```
1. struct Point<T> { x: T, y: T }
```

이제 Point 구조체를 T 타입의 객체로 생성할 수 있다. 이 코드는 3-18번

코드를 3-18번 코드로 복사한다

```
1. #[derive(Debug, PartialEq)]
2. struct Point<T> {x: T, y: T}
3. impl<T> Point<T> {
4.     fn new(x: T, y: T) -> Self{
5.         Point{x: x, y: y}
6.     }
7. }
8. fn main(){
9.     let point1 = Point::new(1, 2);
10.    let point2 = Point::new("1", "2");
11.    assert_eq!(point1, Point{x: 1, y: 2});
12.    assert_eq!(point2, Point{x: "1", y: "2"});
13. }
```

이제 3번 코드를 impl T 블록에서 Rust의 Vec 구조체로 변경한다. 이 코드는 3-19번 코드로 복사한다

이제 3-19번 코드를 Vec T 블록으로 복사한다

```
1. pub struct Vec<T> {
2.     buf: RawVec<T>,
3.     len: usize,
4. }
```

Rust의 Monomorphization은 이 코드를 3-16번 코드로 복사한다. 이 코드는 3-16번 코드로 복사한다. 이 코드는 3-20번

이제 3-20번 코드를 복사한다

```

1.  fn foo_1(x: i32) -> i32 {
2.      return x;
3.  }
4.  fn foo_2(x: &'static str) -> &'static str {
5.      return x;
6.  }
7.  fn main(){
8.      foo_1(1);
9.      foo_2("2");
10. }

```

Rustの関数型プログラミングの基礎
 Rustの関数型プログラミングの基礎
 Rustの関数型プログラミングの基礎

3.3.2 関数型プログラミング

Rustの関数型プログラミングの基礎 3-21
 Rustの関数型プログラミングの基礎

```

1.  #[derive(Debug, PartialEq)]
2.  struct Foo(i32);
3.  #[derive(Debug, PartialEq)]
4.  struct Bar(i32, i32);
5.  trait Inst {
6.      fn new(i: i32) -> Self;
7.  }
8.  impl Inst for Foo {
9.      fn new(i: i32) -> Foo {
10.         Foo(i)
11.     }
12. }
13. impl Inst for Bar {
14.     fn new(i: i32) -> Bar {
15.         Bar(i, i + 10)
16.     }
17. }
18. fn foobar<T: Inst>(i: i32) -> T {
19.     T::new(i)
20. }
21. fn main() {
22.     let f: Foo = foobar(10);
23.     assert_eq!(f, Foo(10));
24.     let b: Bar = foobar(20);
25.     assert_eq!(b, Bar(20, 30));
26. }

```

3-21 Foo Bar Inst trait
new foobar T new

22 foobar Foo Rust
Foo new 24 foobar Bar
Rust Bar new

3.4 trait

trait Rust Rust OOP trait trait

trait Rust Ad-hoc trait 4

-
- trait
-
- trait “”

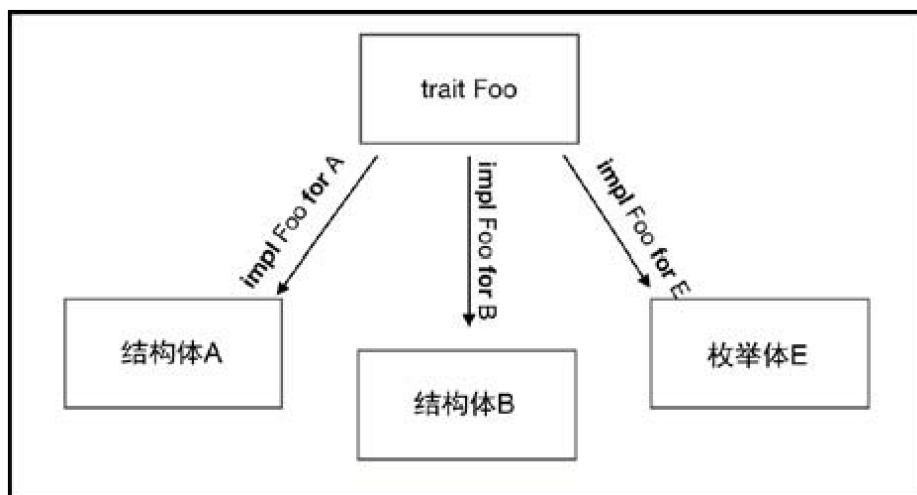
trait 4

3.4.1

trait

-
-
-
- impl
- trait

3-2 trait



3-2 trait

2-53 Fly trait Duck Pig
trait Ad-hoc trait
trait Ad-hoc
trait

Rust trait trait

trait 3-22

3-22 trait

```
1. trait Add<RHS, Output > {
2.     fn add(self, rhs: RHS) -> Output;
3. }
4. impl Add<i32, i32> for i32 {
5.     fn my_add(self, rhs: i32) -> i32 {
6.         self + rhs
7.     }
8. }
9. impl Add<u32, i32> for u32 {
10.    fn my_add(self, rhs: u32) -> i32 {
11.        (self + rhs ) as i32
12.    }
13. }
14. fn main(){
15.    let (a, b, c, d) = (1i32, 2i32, 3u32, 4u32);
16.    let x: i32 = a.my_add(b);
17.    let y: i32 = c.my_add(d);
18.    assert_eq!(x, 3i32);
19.    assert_eq!(y, 7i32);
20. }
```

3-22 Add trait RHS Output
trait add self
trait

```

    impl i32 {
        fn add_u32(self, rhs: u32) -> u32 {
            self + rhs as u32
        }
    }

    impl u32 {
        fn add_i32(self, rhs: i32) -> i32 {
            self + rhs as i32
        }
    }

    fn main() {
        let i32_val = 4;
        let u32_val = 8;

        println!("4 + 8 = {}", i32_val.add_u32(u32_val));
        println!("8 + 4 = {}", u32_val.add_i32(i32_val));
    }

    trait Add {
        type Output;
        fn add(self, rhs: Self) -> Self::Output;
    }

    impl Add for i32 {
        type Output = i32;
        fn add(self, rhs: i32) -> i32 {
            self + rhs
        }
    }

    impl Add for u32 {
        type Output = u32;
        fn add(self, rhs: u32) -> u32 {
            self + rhs
        }
    }

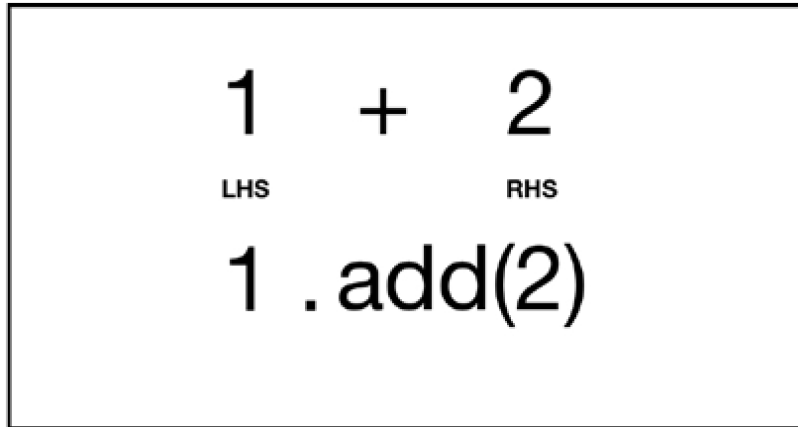
    impl Add for String {
        type Output = String;
        fn add(self, rhs: String) -> String {
            self + rhs
        }
    }

    impl Add for Rust {
        type Output = Rust;
        fn add(self, rhs: Rust) -> Rust {
            self + rhs
        }
    }

    // 3-23 Rust Add trait
    // 3-23 Add trait
    1. pub trait Add<RHS = Self> {
    2.     type Output;
    3.     fn add(self, rhs: RHS) -> Self::Output;
    4. }

    // 3-23 trait 3-22 trait
    // type Output
    // Add
    // RHS=Self
    // RHS
    // Self
    // Self
    // trait
    // trait
    // "+"
    // Rust
    // add
    // "+"
    // add
    // 3-3

```



3-3 "+" 运算符 add

3-24 为 `u32` 实现 `Add` trait

3-24 为 `u32` 实现 `Add` trait

1. `impl Add for $t {`
2. `type Output = $t;`
3. `fn add(self, other: $t) -> $t { self + other }`
4. `}`

Rust 为 `u32` 实现 `Add` trait

12 为 `$t` 实现 `u32`

3-25

3-25 为 `$t` 实现 `u32`

1. `impl Add for u32 {`
2. `type Output = u32;`
3. `fn add(self, other: u32) -> u32 { self + other }`
4. `}`

为 `u32` 实现 `u32` 实现 `Add` trait

Self 为 `u32`

String 实现 `String`

3-26 String

Add trait

3-26 为 `String` 实现 `Add` trait

```

1. impl Add<&str> for String {
2.     type Output = String;
3.     fn add(mut self, other: &str) -> String {
4.         self.push_str(other);
5.         self
6.     }
7. }

```

3-26 `impl Add<&str> for String` `&str` `Self` `String` `&str` `String` `Output` `String` `String` 3-27 `String`

3-27 **String**

```

1. fn main() {
2.     let a = "hello";
3.     let b = " world";
4.     let c = a.to_string() + b;
5.     println!("{:?}", c); // "hello world"
6. }

```

3-27 `a` `b` `&str` `a` `String` **trait** **trait**

trait

`Add` `trait` `impl Add` `Rust` `Add` `trait` `u32` `u64` 3-28 `u32` `u64`

3-28

```

1. use std::ops::Add;
2. impl Add<u64> for u32{
3.     type Output = u64;
4.     fn add(self, other: u64) -> Self::Output {
5.         (self as u64) + other
6.     }
7. }
8. fn main(){
9.     let a = 1u32;
10.    let b = 2u64;
11.    assert_eq!(a+b, 3);
12. }

```

3-28

error[E0117]: only traits defined in the current crate can be implemented for arbitrary types

Rust **Orphan Rule** trait trait trait crate 3-28 Add trait u32 u64 crate u32 crate Bug

Add trait crate 3-29

3-29 crate Add trait

```

1.  trait Add<RHS=Self> {
2.      type Output;
3.      fn add(self, rhs: RHS) -> Self::Output;
4.  }
5.  impl Add<u64> for u32{
6.      type Output = u64;
7.      fn add(self, other: u64) -> Self::Output {
8.          (self as u64) + other
9.      }
10. }
11. fn main(){
12.     let a = 1u32;
13.     let b = 2u64;
14.     assert_eq!(a.add(b), 3);
15. }

```

3-29 crate Add trait impl
 Add RHS u64 add +
 Rust add

Add trait
 Add 3-30

3-30 Add

```

1. use std::ops::Add;
2. #[derive(Debug)]
3. struct Point {
4.     x: i32,
5.     y: i32,
6. }
7. impl Add for Point {
8.     type Output = Point;
9.     fn add(self, other: Point) -> Point {
10.         Point {
11.             x: self.x + other.x,
12.             y: self.y + other.y,
13.         }
14.     }
15. }
16. fn main() {
17.     // Point { x: 3, y: 3 }
18.     println!("{:?}", Point { x: 1, y: 0 } + Point { x: 2, y: 3 });
19. }

```

输出 **Output** 类型 `add` 函数返回 `Point` 类型
`Self` `Self` `Output`

trait

Rust 中的 **trait** 类似于其他语言中的 `interface`
 定义 `trait` 的语法如下：
`trait TraitName {`
 `// 方法定义`
`}`

例如 `Web` 是一个 `trait`，定义如下：
`trait Web {`
 `// 方法定义`
`}`

3-31 **trait**

```

1.  trait Page{
2.      fn set_page(&self, p: i32){
3.          println!("Page Default: 1");
4.      }
5.  }
6.  trait PerPage{
7.      fn set_perpage(&self, num: i32){
8.          println!("Per Page Default: 10");
9.      }
10. }
11. struct MyPaginate{ page: i32 }
12. impl Page for MyPaginate{}
13. impl PerPage for MyPaginate{}
14. fn main(){
15.     let my_paginate = MyPaginate{page: 1};
16.     my_paginate.set_page(2);
17.     my_paginate.set_perpage(100);
18. }

```

3-31 trait Page PerPage trait
 set_page set_perpage
 1 10

11 MyPaginate
 12 13 MyPaginate Page PerPage impl
 trait

14 18 main MyPaginate
 my_paginate set_page set_perpage

trait
 3-32

3-32 trait

3-34 sum

trait

trait

3-34 3-35

3-35

```
1. use std::ops::Add;
2. fn sum<T: Add<T, Output=T>>(a: T, b: T) -> T{
3.     a + b
4. }
5. fn main(){
6.     assert_eq!(sum(1u32, 2u32), 3);
7.     assert_eq!(sum(1u64, 2u64), 3);
8. }
```

3-35 T Add T Output=T sum Add trait Add T Output=T T Add Output=T

sum String String &str sum Add trait

trait trait trait Bound

```
fn generic<T: MyTrait + MyOtherTrait + SomeStandardTrait>(t: T) {}
```

T T MyTrait MyOtherTrait SomeStandardTrait

trait

trait Java Ruby Python Duck Typing Golang Structural Typing Elixir Clojure Protocol trait Structural Typing Duck Typing

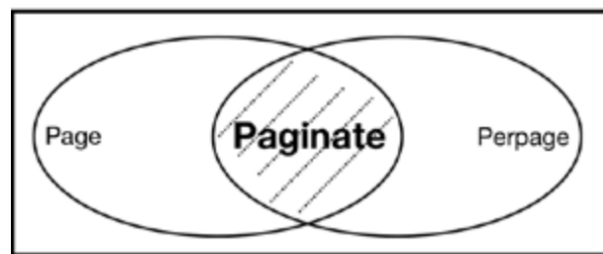
Protocol \rightarrow Structural Typing \rightarrow **Duck Typing**

Rust \rightarrow trait \rightarrow Structural Typing \rightarrow **Duck Typing**

let $x: \text{u32}$ $x \in \text{u32}$ $x: \text{u32}$ \rightarrow trait

```
trait Paginate: Page + PerPage
```

trait \rightarrow $\text{Page} \cap \text{Perpage} \rightarrow \text{Paginate}$

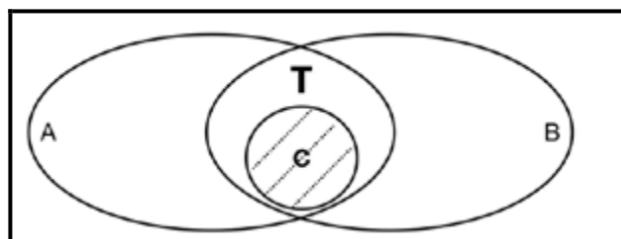


$\text{Page} \cap \text{Perpage} \rightarrow \text{Paginate}$

Rust \rightarrow “ impl ”

```
impl<T: A + B> C for T
```

“ $T \subset A \cap B \rightarrow \text{Trait C}$ ”



$T \subset A \cap B \rightarrow \text{Trait C}$

Rust \rightarrow Rust \rightarrow trait

trait \rightarrow trait

trait 和 where 子句

trait 子句

```
fn foo<T: A, K: B+C, R: D>(a: T, b: K, c: R) { . . . }
```

Rust 中的 **where** 子句

```
fn foo<T, K, R>(a: T, b: K, c: R) where T: A, K: B+C, R: D { . . . }
```

trait 子句

3.4.3 trait 子句

trait 子句 **Abstract Type** 子句

子句 **Existential Type** 子句

子句 Rust 子句

子句 **trait** 子句 **impl Trait** 子句

trait 子句

子句 trait 子句

子句 trait 子句 **trait Object** 子句

“子句”子句 trait 子句

子句 trait 子句

子句 3-36 子句 trait 子句 trait 子句

子句 3-36 **trait** 子句 **trait** 子句

```

1.  #[derive(Debug)]
2.  struct Foo;
3.  trait Bar {
4.      fn baz(&self);
5.  }
6.  impl Bar for Foo {
7.      fn baz(&self) { println!("{:?}", self) }
8.  }
9.  fn static_dispatch<T>(t: &T) where T:Bar {
10.     t.baz();
11. }
12. fn dynamic_dispatch(t: &Bar) {
13.     t.baz();
14. }
15. fn main() {
16.     let foo = Foo;
17.     static_dispatch(&foo);
18.     dynamic_dispatch(&foo);
19. }

```

3-36 Foo Bar trait Foo Bar

9 14 trait static_dispatch trait
dynamic_dispatch

15 19 static_dispatch dynamic_dispatch
static_dispatch t baz Foo Bar
Bar dynamic_dispatch t &Bar trait
trait

trait trait
& Box T trait trait 3-37

3-37 trait

```

1. pub struct TraitObject {
2.     pub data: *mut (),
3.     pub vtable: *mut (),
4. }

```

图3-37展示了TraitObject在Rust中的内存布局。它包含两个指针：data和vtable。data指向一个Unsafe trait对象，vtable指向一个Virtual Table。

TraitObject 包含 **data** 和 **vtable**。在 `impl MyTrait for T` 中，`data` 指向 `trait` 对象，`vtable` 指向 `T` 的 `MyTrait` 的 `Vtable`（Virtual Table）。C++ 中的 `TraitObject` 如图3-6所示。

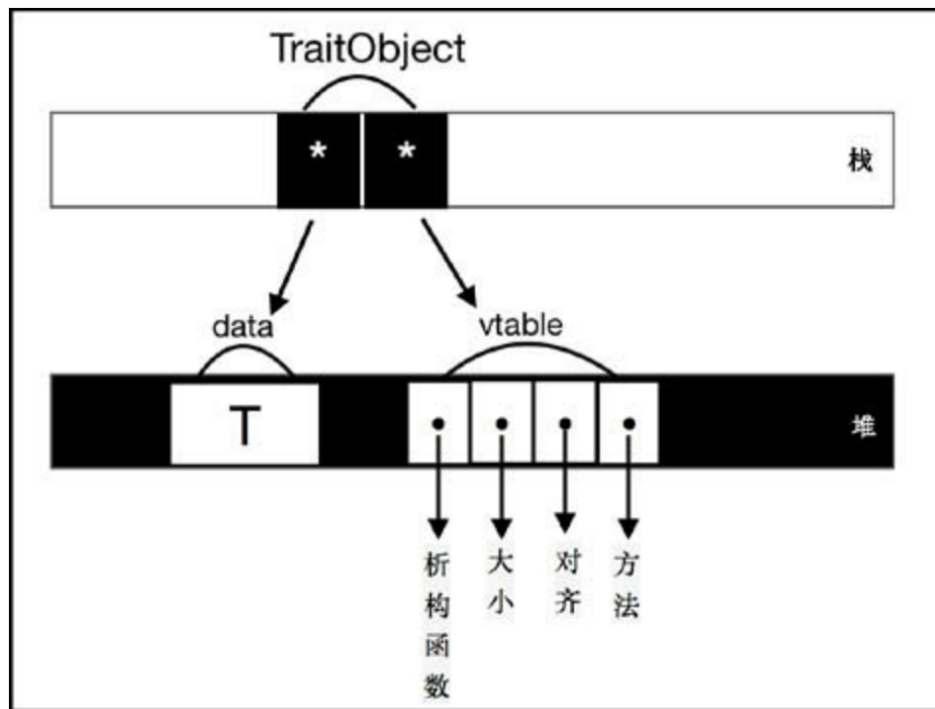


图3-6 TraitObject 内存布局

在 Rust 中，`TraitObject` 的内存布局如下：它包含两个指针：`data` 和 `vtable`。在 `trait_object.method` 中，`TraitObject` 的 `data` 指向 `trait` 对象，`vtable` 指向 `T` 的 `MyTrait` 的 `Vtable`。

图 3-36 展示了 `dynamic_dispatch` 和 `&foo` 的内存布局。在 `t.baz` 中，`trait` 对象指向 `T` 的 `MyTrait` 的 `Vtable`。

在 `trait` 对象中，`data` 指向 `T` 的 `MyTrait` 的 `Vtable`。


```
1. use std::fmt::Debug;
2.     pub trait Fly {
3.         fn fly(&self) -> bool;
4.     }
5.     #[derive(Debug)]
6.     struct Duck;
7.     #[derive(Debug)]
8.     struct Pig;
9.     impl Fly for Duck {
10.         fn fly(&self) -> bool {
11.             return true;
12.         }
13.     }
14.     impl Fly for Pig {
15.         fn fly(&self) -> bool {
16.             return false;
17.         }
18.     }
19.     fn fly_static(s: impl Fly+Debug) -> bool {
20.         s.fly()
21.     }
22.     fn can_fly(s: impl Fly+Debug) -> impl Fly {
23.         if s.fly(){
24.             println!("{:?} can fly", s);
25.         }else{
26.             println!("{:?} can't fly", s);
27.         }
28.         s
29.     }
30.     let pig = Pig;
31.     assert_eq!(fly_static(pig), false);
32.     let duck = Duck;
33.     assert_eq!(fly_static(duck), true);
34.     let pig = Pig;
35.     let pig = can_fly(pig);      // Pig 不能执行“飞”这个动作
36.     let duck = Duck;
37.     let duck = can_fly(duck);    // Duck 能执行“飞”这个动作
38. }
```

3-41 19 21 impl Fly+Debug trait
impl Trait trait

22 29 can_fly impl Fly+Debug
impl Fly **impl Trait** trait

main fly_static turbofish
Rust turbofish
can_fly impl Fly

let impl Fly let duck impl
Fly=can_fly duck trait
impl Trait

impl Trait impl Trait
3-42

3-42 impl Trait

```
1. use std::ops::Add;
2. fn sum<T>(a: impl Add<Output=T>, b: impl Add<Output=T>) -> T{
3.     a + b
4. }
```

3-42 sum a b impl Add
Output=T a b

Rust 2018 impl Trait **trait**
dyn Trait dyn Dynamic impl Trait
dyn Trait

3-42 dyn Trait 3-43

3-43 3-42 dyn Trait

```

    3-433-42dyn_can_flydyn
TraitBoxdyn Flytrait Rust 2015
BoxFlystaticimpl
Fly+Debug5

```

```
trait Empty { }
“Empty” “Empty” trait Rust “Empty”
“Empty” trait “Empty” “Empty”
“Empty” “Empty” “Empty”
```

- **Sized** trait
- **Unsize** trait
- **Copy** trait
- **Send** trait
- **Sync** trait

Sized trait

[illegible]

3-44 Sized trait

```
1. #[lang = "sized"]
2. pub trait Sized {
3.     //代码为空，无具体实现方法
4. }
```

```

    Sized trait trait trait " "
    3-44 1 [lang= sized] lang Sized trait
    Rust sized Lang Item
    Sized trait a+b
    Add add a b [lang= add]

```

```
Rust::Sized::Sized
trait 3-45
```

3-45 Sized trait

```
1. struct Foo<T>(T);
2. struct Bar<T: ?Sized>(T);
```

```

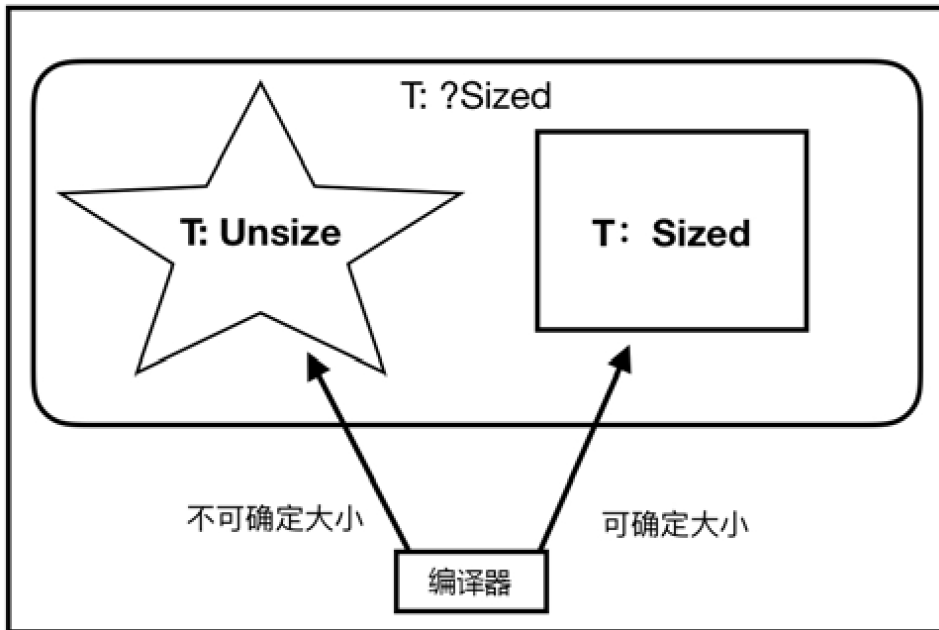
    3-45FooT Sized
    T Sized

```

```

    Sized Sized trait {} {} {} {} Sized Unsize {} {} Sized {} {} {} 3-8
    {} {}

```



3-8 Sized 与 Unsize 与 Sized 的

Sized 与 Unsize 是 Rust 中的两个 trait，它们定义了类型的大小。Sized 是一个 trait，它要求类型的大小是有限的。Unsize 是一个 trait，它要求类型的大小是无限的。在 Rust 中，Sized 是一个 trait，它要求类型的大小是有限的。Unsize 是一个 trait，它要求类型的大小是无限的。在 Rust 中，Sized 是一个 trait，它要求类型的大小是有限的。Unsize 是一个 trait，它要求类型的大小是无限的。

在 Rust 中，Sized 是一个 trait，它要求类型的大小是有限的。Unsize 是一个 trait，它要求类型的大小是无限的。在 Rust 中，Sized 是一个 trait，它要求类型的大小是有限的。Unsize 是一个 trait，它要求类型的大小是无限的。

在 Rust 中，Sized 是一个 trait，它要求类型的大小是有限的。Unsize 是一个 trait，它要求类型的大小是无限的。

- 在 Rust 中，Sized 是一个 trait，它要求类型的大小是有限的。Unsize 是一个 trait，它要求类型的大小是无限的。
- 在 Rust 中，Sized 是一个 trait，它要求类型的大小是有限的。Unsize 是一个 trait，它要求类型的大小是无限的。
- 在 Rust 中，Sized 是一个 trait，它要求类型的大小是有限的。Unsize 是一个 trait，它要求类型的大小是无限的。

在 Rust 中，Sized 是一个 trait，它要求类型的大小是有限的。Unsize 是一个 trait，它要求类型的大小是无限的。

Copy trait

Copy trait 是 Rust 中的一个 trait，它要求类型的大小是有限的。Copy trait 是 Rust 中的一个 trait，它要求类型的大小是有限的。Copy trait 是 Rust 中的一个 trait，它要求类型的大小是有限的。Copy trait 是 Rust 中的一个 trait，它要求类型的大小是有限的。

3-46 Copy trait 的

1. `#[lang = "copy"]`
2. `pub trait Copy : Clone {`
3. `//代码为空，无具体实现方法`
4. `}`

在 Rust 中，Copy trait 是一个 trait，它要求类型的大小是有限的。Copy trait 是一个 trait，它要求类型的大小是有限的。Copy trait 是一个 trait，它要求类型的大小是有限的。Copy trait 是一个 trait，它要求类型的大小是有限的。

3-47 Clone trait 的

1. `pub trait Clone : Sized {`
2. `fn clone(&self) -> Self;`
3. `fn clone_from(&mut self, source: &Self) {`
4. `*self = source.clone();`
5. `}`
6. `}`

Clone trait Sized Clone trait
Sized 3-47 3 clone_from clone Clone trait clone

Copy trait Clone trait 3-48

3-48 Copy trait Clone trait

```
1. struct MyStruct;  
2. impl Copy for MyStruct { }  
3. impl Clone for MyStruct {  
4.     fn clone(&self) -> MyStruct {  
5.         *self  
6.     }  
7. }
```

Rust derive
3-49

3-49 derive Copy trait Clone trait

```
1. #[derive(Copy, Clone)]  
2. struct MyStruct;
```

Rust Copy trait Char
3-50 Copy trait
Copy trait test_copy Copy trait

3-50 Copy trait

```
1. fn test_copy<T: Copy>(i: T) {  
2.     println!("hhh");  
3. }  
4. fn main() {  
5.     let a = "String".to_string();  
6.     test_copy(a);  
7. }
```

3-50 String 的 Copy trait

error[E0277]: the trait bound `std::string::String: std::marker::Copy` is not satisfied

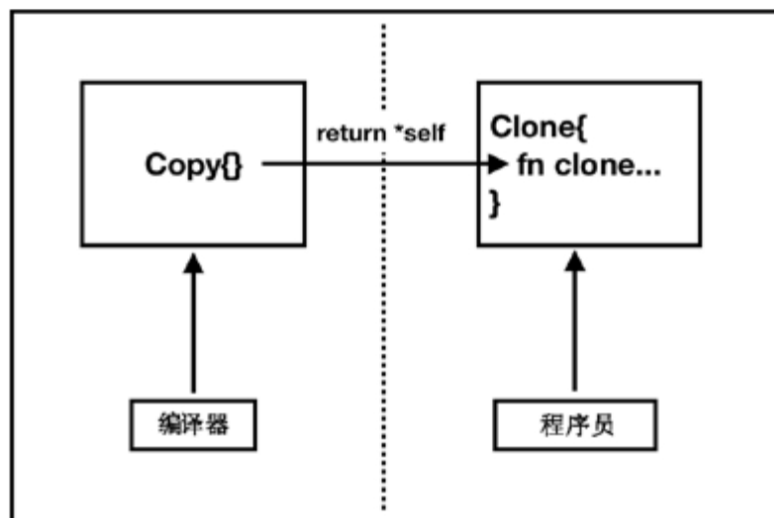
```
| test_copy(a);  
| ^^^^^^^^^ the trait `std::marker::Copy` is not implemented for  
| `std::string::String`
```

String 的 Copy trait

Copy trait 是 Rust 中的一个 trait，它定义了一个 `clone` 方法。任何实现了 `Copy` trait 的类型都可以被复制。在 Rust 中，`Copy` trait 是 `Clone` trait 的子 trait。任何实现了 `Copy` trait 的类型都必须实现 `Clone` trait。在 Rust 中，`Copy` trait 是 `Clone` trait 的子 trait。任何实现了 `Copy` trait 的类型都必须实现 `Clone` trait。在 Rust 中，`Copy` trait 是 `Clone` trait 的子 trait。任何实现了 `Copy` trait 的类型都必须实现 `Clone` trait。

Clone trait 是 Rust 中的一个 trait，它定义了一个 `clone` 方法。任何实现了 `Clone` trait 的类型都可以被复制。在 Rust 中，`Copy` trait 是 `Clone` trait 的子 trait。任何实现了 `Copy` trait 的类型都必须实现 `Clone` trait。在 Rust 中，`Copy` trait 是 `Clone` trait 的子 trait。任何实现了 `Copy` trait 的类型都必须实现 `Clone` trait。

Copy trait 是 Rust 中的一个 trait，它定义了一个 `clone` 方法。任何实现了 `Copy` trait 的类型都可以被复制。在 Rust 中，`Copy` trait 是 `Clone` trait 的子 trait。任何实现了 `Copy` trait 的类型都必须实现 `Clone` trait。在 Rust 中，`Copy` trait 是 `Clone` trait 的子 trait。任何实现了 `Copy` trait 的类型都必须实现 `Clone` trait。



[illegible]

Race Condition

Data Race

"Erlang Actor Golang CSP Rust"

```
Rust trait Send Sync trait Rust
· Send
· Sync
trait Rust
Rust
```

3-51

3-51

```
1. use std::thread;
2. fn main() {
3.     let x = vec![1, 2, 3, 4];
4.     thread::spawn(|| x);
5. }
```

[illegible]

例3-52

```
1. use std::thread;
2. fn main() {
3.     let mut x = vec![1, 2, 3, 4];
4.     thread::spawn(|| {
5.         x.push(1);
6.     });
7.     x.push(2);
8. }
```

例3-52 `x` を `push` する `x` の 5 番目の
要素 `push` する 2

例3-52 のコードは、`x` を `push` する 5 番目の
要素 `push` する 2

例3-51

error[E0373]: closure may outlive the current function, but it borrows
`x`, which is owned by the current function

```
|
|     thread::spawn( || {
|         ^^ may outlive borrowed value `x`
|         x.push(1);
|         - `x` is borrowed here
```

help: to force the closure to take ownership of `x` (and any other referenced
variables), use the `move` keyword, as shown:

```
|     thread::spawn( move || {
```

例3-53 `Rust` の `x` を `move` する `x` の 5 番目の
要素 `move` する 2

例3-53

```

1. use std::thread;
2. fn main() {
3.     let mut x = vec![1, 2, 3, 4];
4.     thread::spawn(move || x.push(1));
5.     // x.push(2);
6. }

```

3-53 5
 `move`
`x`
`Send`
`Sync`
`trait`
`x`
`x`
 3-53 5

3-54 `Send` `Sync`

3-54 `Send` `Sync`

```

1. use std::thread;
2. use std::rc::Rc;
3. fn main() {
4.     let x = Rc::new(vec![1, 2, 3, 4]);
5.     thread::spawn( move || {
6.         x[1];
7.     });
8. }

```

3-54 `std::rc::Rc` `Rc` `Send` `Sync`
`x`

```

error[E0277]: the trait bound `std::rc::Rc<std::vec::Vec<i32>>:
std::marker::Send` is not satisfied in `[closure@src/main.rs:
x:std::rc::Rc<std::vec::Vec<i32>>]`
|     thread::spawn( move || {
|     ^^^^^^^^^^^^^^^ `std::rc::Rc<std::vec::Vec<i32>>` cannot be sent
between threads safely

```

`x`
`std::rc::Rc`
`std::vec::Vec`
`i32`
`Rc`
`Rc`
`x`
`Rc`
`Rc`
`Rust`

SendSync trait 与 Copy Sized 类型的关系
Send Sync 与 SendSync 3-55

3-55 Send Sync

```
1.  #[lang = "send"]
2.  pub unsafe trait Send {
3.      //代码为空，无具体实现方法
4.  }
5.  ...
6.  #[lang = "sync"]
7.  pub unsafe trait Sync {
8.      //代码为空，无具体实现方法
9.  }
```

3-56 Rust Send Sync

3-56 Rust Send Sync

```
1.  unsafe impl Send for .. { }
2.  impl<T: ?Sized> !Send for *const T { }
3.  impl<T: ?Sized> !Send for *mut T { }
```

3-56 1 for.. Send Sync
2 3 Send
3-56 Rust src/libcore/marker.rs

Send Sync Rust Copy Clone derive Send Sync

Rust Send Sync “”

3.5


```

1. fn main() {
2.     let a = "hello".to_string();
3.     let b = " world".to_string();
4.     let c = a + &b;
5.     println!("{:?}", c); // "hello world"
6. }

```

a b String &b
 &String String add &str
 3-58 String Deref
 Target=str 3-59

3-59 String Deref Target=str

```

1. impl ops::Deref for String {
2.     type Target = str;
3.     fn deref(&self) -> &str {
4.         unsafe { str::from_utf8_unchecked(&self.vec) }
5.     }
6. }

```

&String &str 3-58 String
 Deref Vec T Deref 3-
 60 Box T Rc T Arc T **Deref**

3-60 Vec T Deref

```

1. fn foo(s: &[i32]) {
2.     println!("{:?}", s[0]);
3. }

4. fn main() {
5.     let v = vec![1,2,3];
6.     foo(&v)
7. }

```

3-60 foo &[T] foo &v &v
 &Vec T Vec T Deref Target=[T]

`&Vec<T>::new(&[T]::foo)` の型は `Vec<T>` である。

3-61 `Rc` の `Deref` トレイトの実装

3-61 `Rc` の `Deref`

```
1. use std::rc::Rc;
2. fn main() {
3.     let x = Rc::new("hello");
4.     println!("{:?}", x.chars());
5. }
```

3-61 `x` の `Rc` の `&str` トレイトの実装 `chars` の実装
`Rc` の `T` の `Deref` の `Target` の `T` の実装
`Rc` の実装

3-61

`Deref` の実装 3-61 `Rc` の `chars` の実装
3-62

3-62 `clone`

```
1. use std::rc::Rc;
2. fn main() {
3.     let x = Rc::new("hello");
4.     let y = x.clone(); // Rc<&str>
5.     let z = (*x).clone(); // &str
6. }
```

3-62 `clone` の `Rc` の `&str` の実装 `Rc` の `clone` の実装 `Rc` の `&str` の `clone` の実装 “`clone`” の実装

`match` の実装 3-63

3-63 `match`

```

1. fn main() {
2.     let x = "hello".to_string();
3.     match &x {
4.         "hello" => {println!("hello")},
5.         _ => {}
6.     }
7. }

```

3-63 `&String` `&str`

- `match x.deref()` `deref()` `use std::ops::Deref`
- `match x.as_ref()` `String` `as_ref()` `&str` **AsRef** trait

- `match x.borrow()` `borrow()` **Borrow** trait **AsRef** **use std::borrow::Borrow**

- `match &*x` `"` `"` `String` `str` `"` `"` `&str`

- `match &x[..]` `String` `index()` `&str`

`Rust`

3.5.2 as

`as` `Rust` `as`

3-64

3-64 **as**

```

1. fn main() {
2.     let a = 1u32;
3.     let b = a as u64;
4.     let c = 3u64;
5.     let d = c as u32;
6. }

```


3-64 `u32` `u64` `as`
3-65

3-65 `u32` `u16`

```
1. fn main() {  
2.     let a = std::u32::MAX; // 4294967295  
3.     let b = a as u16;  
4.     assert_eq!(b, 65535);  
5.     let e = -1i32;  
6.     let f = e as u32;  
7.     println!("{:?}", e.abs()); // 1  
8.     println!("{:?}", f); // 4294967295  
9. }
```

3-65 `a` `u32` `u16`
`b` `u16`
`as`

`trait` 3-66 `as`

3-66 `trait`

```

1. struct S(i32);
2. trait A {
3.     fn test(&self, i: i32);
4. }
5. trait B {
6.     fn test(&self, i: i32);
7. }
8. impl A for S {
9.     fn test(&self, i: i32) {
10.         println!("From A: {:?}", i);
11.     }
12. }
13. impl B for S {
14.     fn test(&self, i: i32) {
15.         println!("From B: {:?}", i+1);
16.     }
17. }
18. fn main() {
19.     let s = S(1);
20.     A::test(&s, 1);
21.     B::test(&s, 1);
22.     <S as A>::test(&s, 1);
23.     <S as B>::test(&s, 1);
24. }

```

3-66 S A B trait test

• 3-66 20 21 trait A test B test

• as S as A test S as B test

Fully Qualified Syntax for Disambiguation UFCS S s test S

as A test S A test
S trait

as Rust
&static str &a str
a static a &str
static &str as &static str
&a str 3-67

3-67 as

```
1. fn main(){  
2.     let a: &'static str = "hello"; // &'static str  
3.     let b: &str = a as &str; // &str  
4.     let c: &'static str = b as &'static str; // &'static str  
5. }
```

3-67 as &static str &a str

3.5.3 From Into

From Into std convert trait **from into** 3-68 trait

3-68 **From Into**

```
1. pub trait From<T> {  
2.     fn from(T) -> Self;  
3. }  
4. pub trait Into<T> {  
5.     fn into(self) -> T;  
6. }
```

T From U T from u T
u U 3-69 String from

3-69 **String from**

```

1. fn main(){
2.     let string = "hello".to_string();
3.     let other_string = String::from("hello");
4.     assert_eq!(string, other_string);
5. }

```

3-70 `String` into `U`

3-70 `into`

```

1. #[derive(Debug)]
2. struct Person{ name: String }
3. impl Person {
4.     fn new<T: Into<String>>(name: T) -> Person {
5.         Person {name: name.into()}
6.     }
7. }
8. fn main(){
9.     let person = Person::new("Alex");
10.    let person = Person::new("Alex".to_string());
11.    println!("{:?}", person);
12. }

```

3-70 4 `new` `&str` `String`

Into `U` `From` `T` `into`

3-71 `From` `T` `Into` `U`

```

impl<T, U> Into<U> for T where U: From<T>

```

3-72 `String` `&str`

3-72 `into` `&str` `String`

String From &str into &str String
3-10 From Into



```

trait AsRef, AsMut trait
AsRef Borrow trait AsRef
Borrow trait

```

[illegible]

- 不可变引用
 - 可变引用
- 不可变引用

3.6.1 不可变引用

不可变引用 trait 不可变引用 trait
 trait 不可变引用 trait 不可变引用 trait 不可变引用 trait 不可变引用 trait
 不可变引用 T & a T trait 不可变引用 3-73

不可变 3-73 不可变 T & a T Bar trait

1. impl<T:Foo> Bar for T { }
2. impl<'a,T:Bar> Bar for &'a T { }

不可变 crate 不可变 NewType 不可变
 不可变

不可变 Rc T Option T 不可变
 不可变 3-74 不可变

不可变 3-74 Option T 不可变

```

1. use std::ops::Add;
2. #[derive(PartialEq)]
3. struct Int(i32);
4. impl Add<i32> for Int {
5.     type Output = i32;
6.     fn add(self, other: i32) -> Self::Output {
7.         (self.0) + other
8.     }
9. }
10. // impl Add<i32> for Option<Int> {
11. //     // TODO
12. // }
13. impl Add<i32> for Box<Int> {
14.     type Output = i32;
15.     fn add(self, other: i32) -> Self::Output {
16.         (self.0) + other
17.     }
18. }
19. fn main() {
20.     assert_eq!(Int(3) + 3, 6);
21.     assert_eq!(Box::new(Int(3)) + 3, 6);
22. }

```

3-74 Int Add trait Add trait
Int

Option Int Add 10
12

Box Int Add 20 21

Box T Rust 3-74
crate Box Int trait crate
Box T crate

Box T Rust
[fundamental] Box T 3-75

3-75 Box T

1. `#[fundamental]`
2. `pub struct Box<T: ?Sized>(Unique<T>);`

3-75 Box T **[fundamental]** Box T “”

Box T Fn FnMut FnOnce Sized **[fundamental]** trait Rust

3.6.2

Rust **Overlap** trait 3-76

3-76

1. `impl<T> AnyTrait for T {...}`
2. `impl<T> AnyTrait for T where T: Copy {...}`
3. `impl<T> AnyTrait for i32 {...}`

3-76 AnyTrait

· T
· T where T Copy trait T Copy T

· i32

T T Copy T Copy i32 trait Rust **Blanket Impl**

trait

-
-

3-77

Figure 3-77: Implementing `AddAssign` for `T`

```
1. impl<R, T: Add<R> + Clone> AddAssign<R> for T {
2.     fn add_assign(&mut self, rhs: R) {
3.         let tmp = self.clone() + rhs;

4.         *self = tmp;
5.     }
6. }
```

Figure 3-77 shows the implementation of the `AddAssign` trait for a type `T`. The `add_assign` method takes a mutable reference to `self` and a value `rhs` of type `R`. It clones `self` to create a temporary `tmp`, adds `rhs` to it using the `+` operator, and then assigns the result back to `self` using `*self = tmp`. This ensures that `self` is updated with the sum of its original value and `rhs`.

Figure 3-77 shows the implementation of the `AddAssign` trait for a type `T`. The `add_assign` method takes a mutable reference to `self` and a value `rhs` of type `R`. It clones `self` to create a temporary `tmp`, adds `rhs` to it using the `+` operator, and then assigns the result back to `self` using `*self = tmp`. This ensures that `self` is updated with the sum of its original value and `rhs`.

Figure 3-77 shows the implementation of the `AddAssign` trait for a type `T`. The `add_assign` method takes a mutable reference to `self` and a value `rhs` of type `R`. It clones `self` to create a temporary `tmp`, adds `rhs` to it using the `+` operator, and then assigns the result back to `self` using `*self = tmp`. This ensures that `self` is updated with the sum of its original value and `rhs`.

Figure 3-78 shows the implementation of the `AddAssign` trait for a type `T`. The `add_assign` method takes a mutable reference to `self` and a value `rhs` of type `R`. It clones `self` to create a temporary `tmp`, adds `rhs` to it using the `+` operator, and then assigns the result back to `self` using `*self = tmp`. This ensures that `self` is updated with the sum of its original value and `rhs`.

Figure 3-78 shows the implementation of the `AddAssign` trait for a type `T`. The `add_assign` method takes a mutable reference to `self` and a value `rhs` of type `R`. It clones `self` to create a temporary `tmp`, adds `rhs` to it using the `+` operator, and then assigns the result back to `self` using `*self = tmp`. This ensures that `self` is updated with the sum of its original value and `rhs`.

```

1.  #![feature(specialization)]
2.  struct Diver<T> {
3.      inner: T,
4.  }
5.  trait Swimmer {
6.      fn swim(&self) {
7.          println!("swimming")
8.      }
9.  }
10. impl<T> Swimmer for Diver<T> {}
11. impl Swimmer for Diver<&'static str> {
12.     fn swim(&self) {
13.         println!("drowning, help!")
14.     }
15. }
16. fn main(){
17.     let x = Diver:::<&'static str> { inner: "Bob" };
18.     x.swim(); // drowning, help!
19.     let y = Diver:::<String> { inner: String::from("Alice") };
20.     y.swim(); // swimming
21. }

```

3-78 Diver<T> trait Swimmer trait Diver<T> trait 10

11 15 Diver<&'static str> Swimmer
 main Diver<&'static str> Diver<String>
 swim

Diver<String> " "
 Diver<T> Diver<&'static str> swim

3-78 trait trait
 3-79

3-79 trait

```

1. // 其他代码不变
2. trait Swimmer {
3.     fn swim(&self);
4. }
5. impl<T> Swimmer for Diver<T> {
6.     default fn swim(&self) {
7.         println!("swimming")
8.     }
9. }
10. //其他代码不变

```

图3-79和图3-78展示了Swimmer trait的实现。Diver<T>实现了Swimmer trait的swim方法，并使用了default关键字。图3-78展示了Swimmer trait的定义。

图3-79展示了Swimmer trait的实现。Diver<T>实现了Swimmer trait的swim方法，并使用了default关键字。图3-78展示了Swimmer trait的定义。

图3-79展示了Swimmer trait的实现。Diver<T>实现了Swimmer trait的swim方法，并使用了default关键字。图3-78展示了Swimmer trait的定义。

3.6.3 泛型 trait

Rust的泛型 trait允许我们定义一个 trait，该 trait 可以应用于任何类型。例如，我们可以定义一个 trait，该 trait 可以应用于任何类型，该 trait 可以应用于任何类型。

Rust的泛型 trait允许我们定义一个 trait，该 trait 可以应用于任何类型。例如，我们可以定义一个 trait，该 trait 可以应用于任何类型。

Rust的泛型 trait允许我们定义一个 trait，该 trait 可以应用于任何类型。例如，我们可以定义一个 trait，该 trait 可以应用于任何类型。

图3-80展示了GAT trait的定义。

```

1.  trait StreamingIterator {
2.      type Item<'a>;
3.      fn next<'a>(&'a mut self) -> Option<Self::Item<'a>>;
4.  }

```

3-80 StreamingIterator
 Item a a
 std::io::Lines StreamingIterator

Item a Vec T
 Vec i32 **GAT** **ACT** **Associated type constructor**

GAT Rust

3.7

Rust
 Rust
 Rust **Mental Model**
 Rust

Rust
 Rust
 Rust

Rust “” trait Rust Ad-hoc trait
 trait
 trait impl Trait

Rust
 Rust
 as
 AsRef From/Into trait

[1] [Rust による Rust による Rust](#)

[2] [C の memcpy と Rust の n](#)

[3] [RFC https://github.com/rust-lang/rfcs/blob/master/text/1598-generic_associated_types.md](https://github.com/rust-lang/rfcs/blob/master/text/1598-generic_associated_types.md)

第4章 内存管理

内存管理是操作系统中最基础、最重要的功能之一。

在操作系统中，内存管理的主要任务是：根据用户程序的需要，合理地分配和回收内存空间，以保证系统的高效运行。在C语言中，程序员需要手动管理内存，使用malloc和free函数进行内存的分配和释放。而在Java、C#、Ruby、Python等语言中，垃圾回收（Garbage Collection）机制会自动管理内存，程序员无需手动干预。

然而，内存管理也存在一些常见的Bug，如内存泄漏（Memory Leak）和悬挂指针（Dangling Pointer）。在C/C++中，程序员需要特别注意这些问题，因为它们可能导致程序运行不稳定甚至崩溃。而在垃圾回收语言中，这些问题通常由垃圾回收器自动处理，但并非完全不存在。

GC（垃圾回收）是自动内存管理的一种技术，它通过跟踪内存中的对象使用情况，自动回收不再使用的内存。GC可以分为多种类型，如标记-清除（Mark-Sweep）、复制（Copying）等。在垃圾回收语言中，程序员通常不需要关心内存的分配和释放，但了解GC的原理有助于更好地使用这些语言。

在C/C++中，程序员需要手动管理内存，这虽然增加了程序的复杂性，但也提供了更高的灵活性和性能。而在Java、Python、Ruby等语言中，垃圾回收机制简化了内存管理，提高了开发效率，但可能会带来一定的性能开销。

垃圾回收语言（如Java、Python、Ruby）的GC机制通常会自动处理内存泄漏和悬挂指针问题，但程序员仍需了解其原理。Rust语言则提供了一种新的内存管理方式，它通过编译时检查来避免内存泄漏和悬挂指针，同时保持了C/C++的高性能。

4.1 内存管理

内存管理是操作系统中最基础、最重要的功能之一。在操作系统中，内存管理的主要任务是：根据用户程序的需要，合理地分配和回收内存空间，以保证系统的高效运行。

操作系统在物理内存的基础上，通过虚拟地址空间，为每个进程提供一块连续的、私有的、可寻址的内存空间。

在32位Linux系统中，虚拟地址空间的范围是0x00000000~0xFFFFFFFF，其中0x00000000~0xC0000000是内核空间，0xC0000000~0xFFFFFFFF是用户空间。在64位Linux系统中，虚拟地址空间的范围是0x00000000~0xFFFFFFFF，其中0x00000000~0x00000000是内核空间，0x00000000~0xFFFFFFFF是用户空间。在Windows系统中，虚拟地址空间的范围是0x00000000~0xFFFFFFFF，其中0x00000000~0x7FFFFFFF是用户空间，0x80000000~0xFFFFFFFF是内核空间。

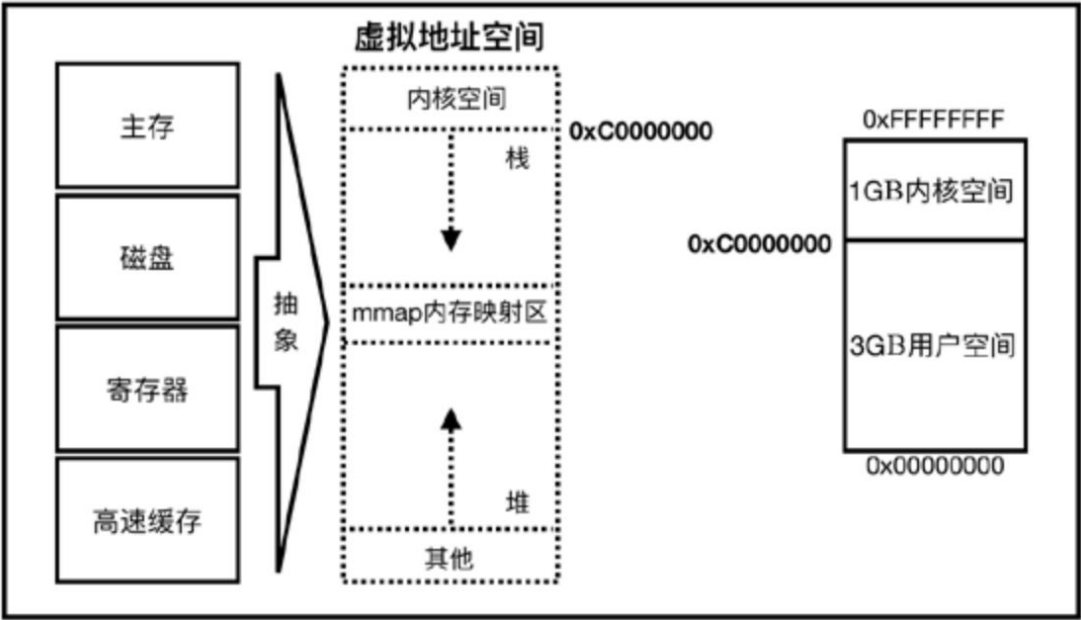


图4-1 Linux虚拟地址空间

在Linux系统中，虚拟地址空间的范围是0x00000000~0xFFFFFFFF，其中0x00000000~0xC0000000是内核空间，0xC0000000~0xFFFFFFFF是用户空间。在64位Linux系统中，虚拟地址空间的范围是0x00000000~0xFFFFFFFF，其中0x00000000~0x00000000是内核空间，0x00000000~0xFFFFFFFF是用户空间。在Windows系统中，虚拟地址空间的范围是0x00000000~0xFFFFFFFF，其中0x00000000~0x7FFFFFFF是用户空间，0x80000000~0xFFFFFFFF是内核空间。

4.1.1

stack是操作系统为用户进程提供的一块连续的、私有的、可寻址的内存空间。

图4-2展示了stack在内存中的分布情况。

图4-2展示了栈的基本操作。栈是一种后进先出（LIFO, Last in First Out）的数据结构。图中显示了入栈（push）和出栈（pop）操作。栈顶（栈顶）指向当前栈顶元素 A_n ，栈底（栈底）指向栈底元素 A_1 。栈中元素按顺序排列，从 A_1 到 A_n 。

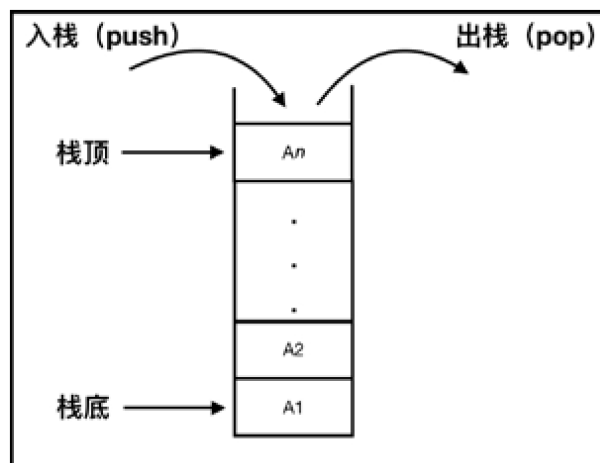


图4-2 栈结构

在操作系统中，栈是由CPU管理的。CPU通过栈顶指针（ESP）来管理栈。图中展示了虚拟地址空间中的栈结构，包括内核空间、mmap内存映射区、堆和其他区域。栈底（栈底）位于高地址处，栈顶（栈顶 ESP）位于低地址处。图中还显示了入栈（push）和出栈（pop）操作的方向。

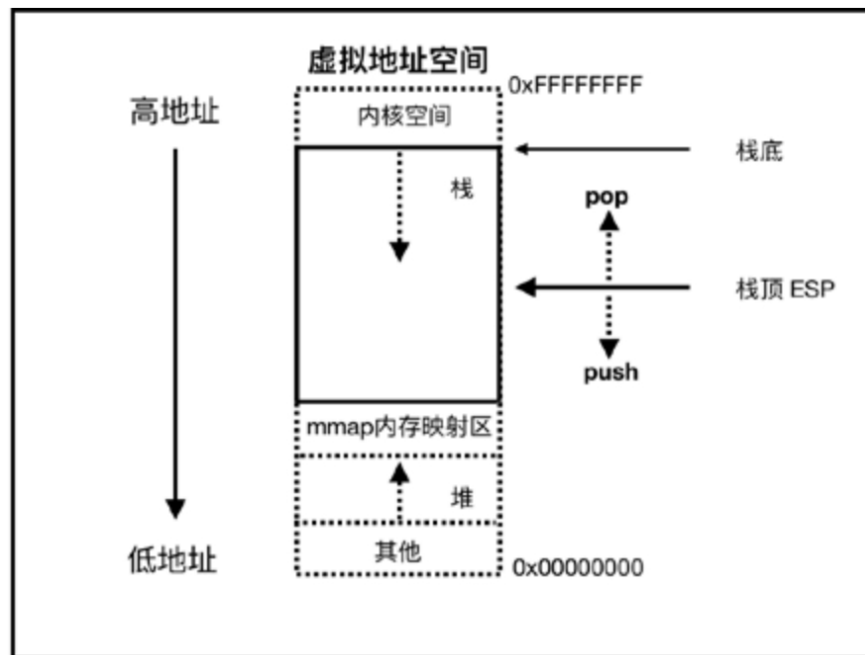
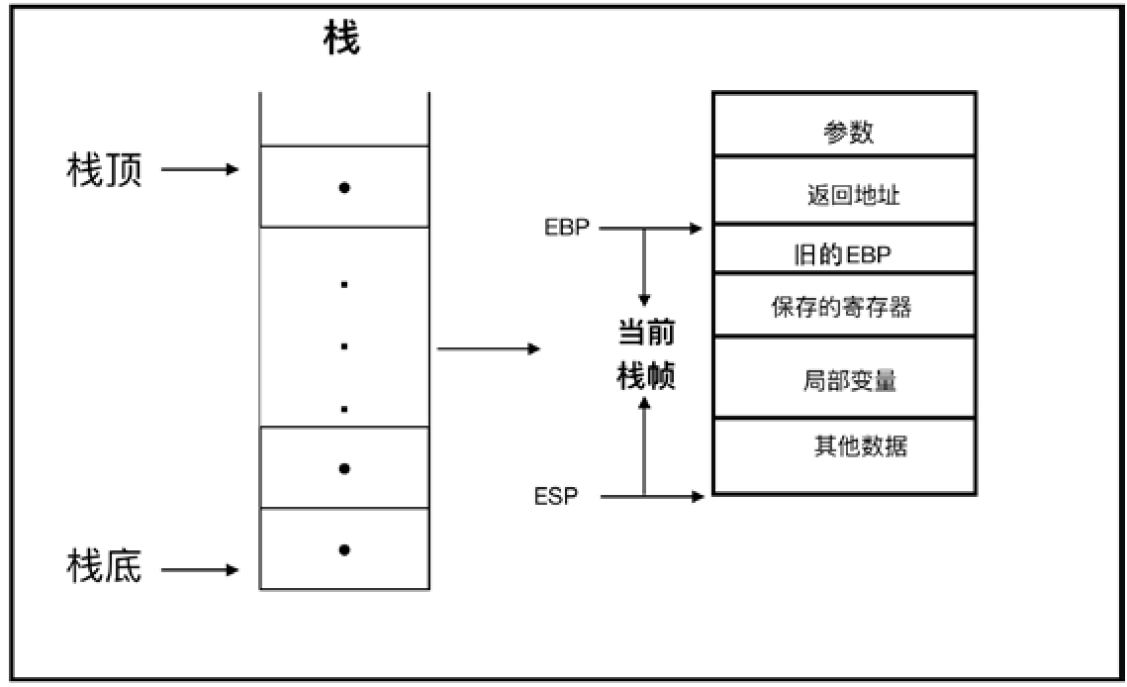


图4-3 栈内存管理

在操作系统中，ESP（栈顶指针）用于管理栈。图中展示了ESP在栈顶的位置，以及入栈（push）和出栈（pop）操作对ESP的影响。ESP指向当前栈顶元素，随着入栈操作，ESP会向低地址方向移动；随着出栈操作，ESP会向高地址方向移动。

ESP

Stack Frame 4-4 Activate Record N



4-4

-
-
-
-

EBP Frame Pointer ESP
EBP EBP
EBP ESP

4-1
foo main x y z

4-1

```

00000004-10000000 main 0000000000000000 main00000000x0000
foo00000000main0000000000000000x00000000EBP00000000x0000
00EBP-4000000000000000

```

The diagram illustrates the stack frame structure before and after calling the `foo` function.

调用foo函数前 (Before calling foo function):

- main 栈帧 (main stack frame):**
 - 初始 EBP (Initial EBP):** Points to address 0.
 - 内容 (Content):** `x = 42`.
 - ESP (Stack Pointer):** Points to address -4.

调用foo函数后 (After calling foo function):

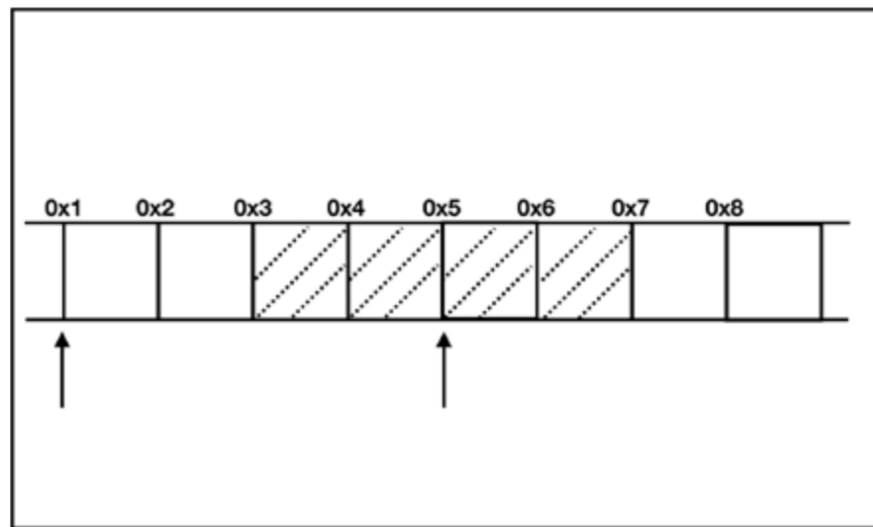
- main 栈帧 (main stack frame):**
 - 初始 EBP (Initial EBP):** Points to address 0.
 - 内容 (Content):** `x = 42` (at +8) and **返回地址 (Return address)** (at +4).
- foo 栈帧 (foo stack frame):**
 - 保存main函数的EBP (Save EBP of main function):** Points to address 0.
 - 内容 (Content):** `x` (at -4), `y` (at -8), and `z` (at -12).
 - ESP (Stack Pointer):** Points to address -12.
- 入栈 (Push):** Indicated by an upward arrow at the bottom.

□4-5□□□□□□□□□□

CPU 32
CPU 32 4 64
CPU 32 CPU

32 CPU CPU 4 4

0x3 CPU 0x7 0x5 4-6 CPU



□4-6□CPU□□□□□4□□□□□□

4-6 32 CPU
4 4-7 CPU

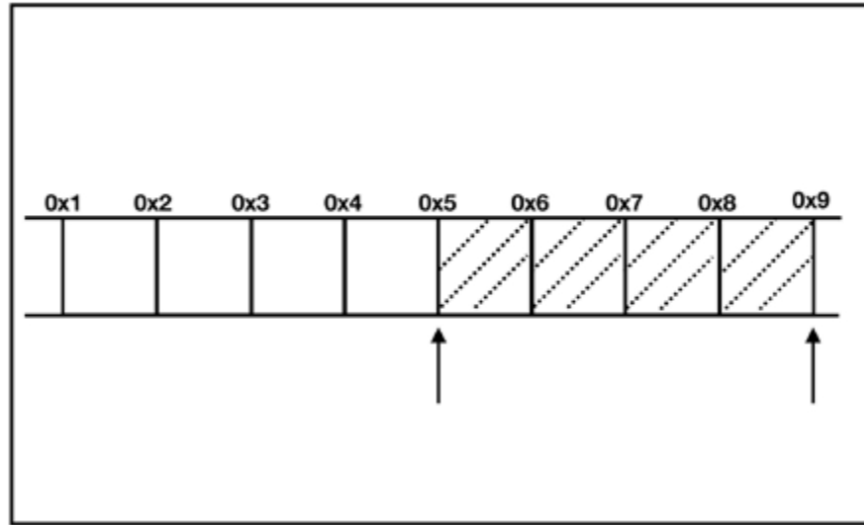


図4-7 CPUメモリアレイ

この図は、CPUメモリアレイの構成を示しています。0x1から0x4までは空欄、0x5から0x9までは斜線で埋められています。JVMのメモリアレイは、このように構成されています。

この図は、CPUメモリアレイの構成を示しています。0x1から0x4までは空欄、0x5から0x9までは斜線で埋められています。JVMのメモリアレイは、このように構成されています。Rustのu32は4バイトで、4つの4バイトのメモリアレイで構成されています。

この図は、CPUメモリアレイの構成を示しています。0x1から0x4までは空欄、0x5から0x9までは斜線で埋められています。JVMのメモリアレイは、このように構成されています。Nは1, 2, 4, 8, 16のいずれかであり、LenはMax LenとNの最小値で決まります。

- ・ Max Lenは、メモリアレイの最大長さを示します。
- ・ Min N Lenは、メモリアレイの最小長さを示します。

- ・ Min N Max Lenは、メモリアレイの最小長さを示します。

この図は、CPUメモリアレイの構成を示しています。0x1から0x4までは空欄、0x5から0x9までは斜線で埋められています。JVMのメモリアレイは、このように構成されています。Rustのu32は4バイトで、4つの4バイトのメモリアレイで構成されています。

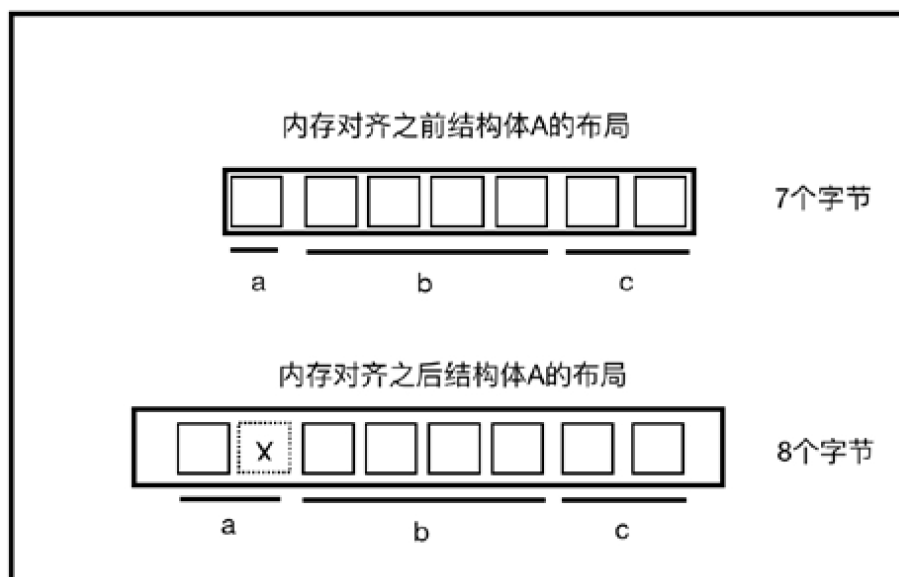
この図は、CPUメモリアレイの構成を示しています。0x1から0x4までは空欄、0x5から0x9までは斜線で埋められています。JVMのメモリアレイは、このように構成されています。Rustのu32は4バイトで、4つの4バイトのメモリアレイで構成されています。

```

1. struct A {
2.     a: u8,
3.     b: u32,
4.     c: u16,
5. }
6. fn main() {
7.     println!("{:?}", std::mem::size_of::<A>()); // 8
8. }

```

4-2 std::mem::size_of A 7 字节
 u8 1 u32 4 u16 2 A 7 字节
 4-8



4-8 A 8 字节

4-8 8 字节 A 7 字节
 4-2 A 7 字节
 b 4 a Min 4 1 1 a
 4-8 x a 2 b c
 A a b c 8 Min 4 4 4
 c A 8

Union
 8

4-3 Rust 编译选项

4-3 Rust 编译选项

```
1. union U {
2.     f1: u32,
3.     f2: f32,
4.     f3: f64
5. }
6. fn main() {
7.     println!("{:?}", std::mem::size_of::<U>()); // 8
8. }
```

4-3 中 **f1** **f2** 为 4 字节 **f3** 为 8 字节 **f3** 占满整个 **U** 8 字节 **f1** **f2** **f3** 均为 8 字节

4.2 Rust 编译选项

Rust 编译选项与 C/C++ 编译选项类似，但 Rust 编译选项中没有 `malloc/new/free/delete` 等选项，Rust 编译选项与 C/C++ 编译选项类似，但 Rust 编译选项中没有 `malloc/new/free/delete` 等选项，Rust 编译选项与 C/C++ 编译选项类似，但 Rust 编译选项中没有 `malloc/new/free/delete` 等选项。

4.2.1 编译选项

2 个编译选项：编译选项 1 为 `rustc`，编译选项 2 为 `rustc`。

const 编译选项：编译选项 1 为 `rustc`，编译选项 2 为 `rustc`。

static 编译选项：编译选项 1 为 `rustc`，编译选项 2 为 `rustc`。

编译选项 1 为 `rustc`，编译选项 2 为 `rustc`。


```
1. fn main() {
2.     let x: i32;
3.     println!("{}", x);
4. }
```

4-4

```
error: use of possibly uninitialized variable: `x`
    println!("{}", x);
```

```
RustXXXXXXXXXXXXXXXXXXXXX4-4XXmainXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXRustXXX
XX
```

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □

Rust ☐ 4-5 ☐ if ☐

4-5 if

```
1.  fn main() {
2.      let x: i32;
3.      if true {
4.          x = 1;
5.      } else {
6.          x = 2;
7.      }
8.      println!("{}", x);
9.  }
```

```

    4-5 if x
    else

```

```
error: use of possibly uninitialized variable: `x`
println!("{}", x);
```

else
if
println
x
2
if
true

4-5
else
8
println
if
x
x
if

break
4-6
break

4-6 loop break

```
1. fn main() {  
2.     let x: i32;  
3.     loop {  
4.         if true {  
5.             x = 2;  
6.             break;  
7.         }  
8.     }  
9.     println!("{}", x); // 2  
10. }
```

Rust
break
x
loop
println
x

4-7

4-7

```
1. fn main() {  
2.     let a: Vec<i32> = vec![];  
3.     let b: [i32; 0] = [];  
4. }
```

error[E0282]: type annotations needed

例4-10 `Drop` trait の実装

例4-11 `Drop` の実装

例4-11 `Drop` の実装

```
1. use std::ops::Drop;
2. #[derive(Debug)]
3. struct S(i32);
4. impl Drop for S {
5.     fn drop(&mut self) {
6.         println!("drop {}", self.0);
7.     }
8. }
9. fn main() {
10.    let x = S(1);
11.    println!("crate x: {:?}", x);
12.    {
13.        let y = S(2);
14.        println!("crate y: {:?}", y);
15.        println!("exit inner scope");
16.    }
17.    println!("exit main");
18. }
```

例4-11 `S` の `impl Drop` の実装

例4-11 `Drop` の実装

例4-12 `Drop` の実装

```

crate x: S(1)
crate y: S(2)
exit inner scope
drop 2
exit main
drop 1

```

例4-11 `x` `main` `y` `scope`
 例4-12 `x` `y` `drop` `RAII`
 例4-13 **Scope-Bound Resource Management (SBRM)**

Drop `drop`

Rust `Vec` `String` `File` `Drop`
`Vec` `String` `File` `Rust`

Valgrind

例4-13 `Box` `T`
Valgrind

4-13 `Box` `T`

```

1. fn create_box() {
2.     let box3 = Box::new(3);
3. }
4. fn main() {
5.     let box1 = Box::new(1);
6.     {
7.         let box2 = Box::new(2);
8.     }
9.     for _ in 0..1_000 {
10.        create_box();
11.    }
12. }

```

例4-13 `box.rs` `rustc` `box`

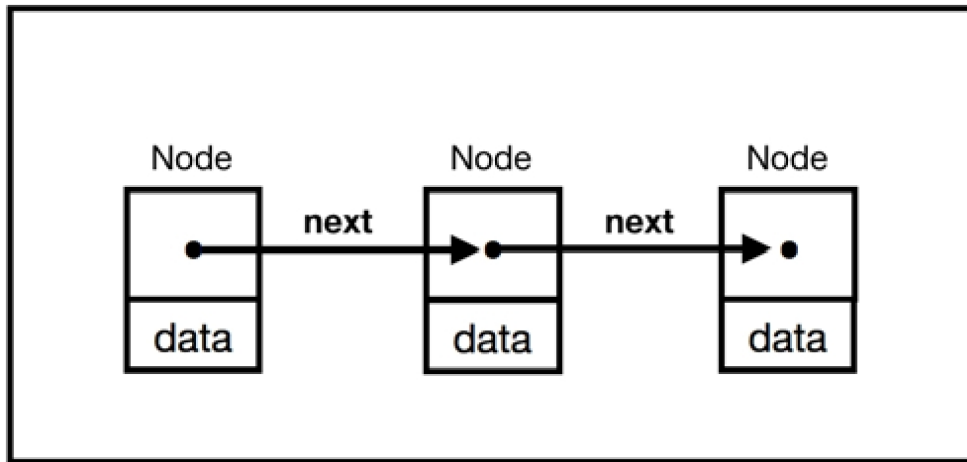
[illegible]

```
1. use std::ops::Drop;
2. #[derive(Debug)]
3. struct S(i32);
4. impl Drop for S {
5.     fn drop(&mut self) {
6.         println!("drop for {}", self.0);
7.     }
8. }
9. fn main() {
10.    let x = S(1);
11.    println!("create x: {:?}", x);
12.    let x = S(2);
13.    println!("create shadowing x: {:?}", x);
14. }
```

4.2.3 □□□□□□□□

RAII 垃圾回收 RAI “ ”

4-10



4-10

Node 4-16

4-16 Node

```
1. struct Node<T> {
2.     data: T,
3.     next: NodePtr<T>,
4. }
```

NodePtr<T>

```
type NodePtr<T> = Option<Box<Node<T>>>
node1.next = node2
node2.next = node3
```

NodePtr<T> Option<T> Some<T> None NodePtr<T> Option<Box<Node<T>>>

node1.next node2.next node1 node2 node3 4-10 Box<T> 4-17 Box<T>

4-17 Box<T>

```

1. type NodePtr<T> = Option<Box<Node<T>>>;
2. struct Node<T> {
3.     data: T,
4.     next: NodePtr<T>,
5. }
6. fn main() {
7.     let mut first = Box::new (Node { data: 1, next : None });
8.     let mut second = Box::new (Node { data: 2, next : None });
9.     first.next = Some(second);
10.    second.next = Some(first);
11. }

```

4-17

error[E0382]: use of moved value: `second`

```

|     first.next = Some(second);
|                               ----- value moved here
|     second.next = Some(first);
|     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ value used here after move

```

4-17 9 second first sencond Box T
second 10

Rust Rc T reference
counting Box T Rc T
NodePtr T Option Rc Node T Rc T
T second.next=Some first
second next Rc T 4-18

4-18 Rc T

```

1. use std::rc::Rc;
2. type NodePtr<T> = Option<Rc<Node<T>>>;
3. struct Node<T> {
4.     data: T,
5.     next: NodePtr<T>,
6. }
7. fn main() {
8.     let first = Rc::new(Node { data: 1, next: None});
9.     let second = Rc::new(Node { data: 2, next: Some(first.clone()) });
10.    first.next = Some(second.clone());
11.    second.next = Some(first.clone());
12. }

```

4-18 first second clone Rc T
clone 1 main

4-18

error[E0594]: cannot assign to immutable field

| first.next = Some(second);

| ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ cannot mutably borrow immutable field

Rust RefCell T
RefCell T 4-18 NodePtr T Option
Rc RefCell Node T 4-19

4-19 RefCell T

```

1. use std::rc::Rc;
2. use std::cell::RefCell;
3. type NodePtr<T> = Option<Rc<RefCell<Node<T>>>>;
4. struct Node<T> {
5.     data: T,
6.     next: NodePtr<T>,
7. }
8. fn main() {
9.     let first = Rc::new(RefCell::new(Node {
10.         data: 1,
11.         next: None,
12.     }));

13.     let second = Rc::new(RefCell::new(Node {
14.         data: 2,
15.         next: Some(first.clone()),
16.     }));
17.     first.borrow_mut().next = Some(second.clone());
18.     second.borrow_mut().next = Some(first.clone());
19. }

```

例 4-19 使用 `Rc<T>` 和 `RefCell<T>` 实现双向链表
 `Node` 的 `Drop` 方法
 例 4-20

例 4-20 `Node` 的 `Drop`

```

1. use std::rc::Rc;
2. use std::cell::RefCell;
3. type NodePtr<T> = Option<Rc<RefCell<Node<T>>>>;
4. struct Node<T> {
5.     data: T,
6.     next: NodePtr<T>,
7. }
8. impl<T> Drop for Node<T> {
9.     fn drop(&mut self) {
10.         println!("Dropping!");
11.     }
12. }
13. fn main() {
14.     let first = Rc::new(RefCell::new(Node {
15.         data: 1,
16.         next: None,
17.     }));
18.     let second = Rc::new(RefCell::new(Node {
19.         data: 2,
20.         next: Some(first.clone()),
21.     }));
22.     first.borrow_mut().next = Some(second.clone());
23.     second.borrow_mut().next = Some(first.clone());
24. }

```

4-20 Node<T> Drop drop
 22 23 first second
 first second

Rust
 Rust

Rust

Rust Rust
 Memory Leak Memory Safety
 Rust

4.2.4 Rust 中的 Enum 和 Union

Rust 中的 Enum 和 Union 是两种特殊的数据类型，用于表示枚举和联合体。

在 Rust 中，Enum 和 Union 的定义和使用如下：

4-22 Rust 中的 Enum 和 Union 的定义

```
1. struct A {
2.     a: u32,
3.     b: Box<u64>,
4. }
5. struct B(i32, f64, char);
6. struct N;
7. enum E {
8.     H(u32),
9.     M(Box<u32>)
10. }
11. union U {
12.     u: u32,
13.     v: u64
14. }
15. fn main(){
16.     println!("Box<u32>: {:?}", std::mem::size_of::<Box<u32>>());
17.     println!("A: {:?}", std::mem::size_of::<A>());
18.     println!("B: {:?}", std::mem::size_of::<B>());
19.     println!("N: {:?}", std::mem::size_of::<N>());
20.     println!("E: {:?}", std::mem::size_of::<E>());
21.     println!("U: {:?}", std::mem::size_of::<U>());
22. }
```

4-22 Rust 中的 Enum 和 Union 的定义

在 Rust 中，Enum 和 Union 的定义和使用如下：

4-11 Rust 中的 Enum 和 Union 的定义

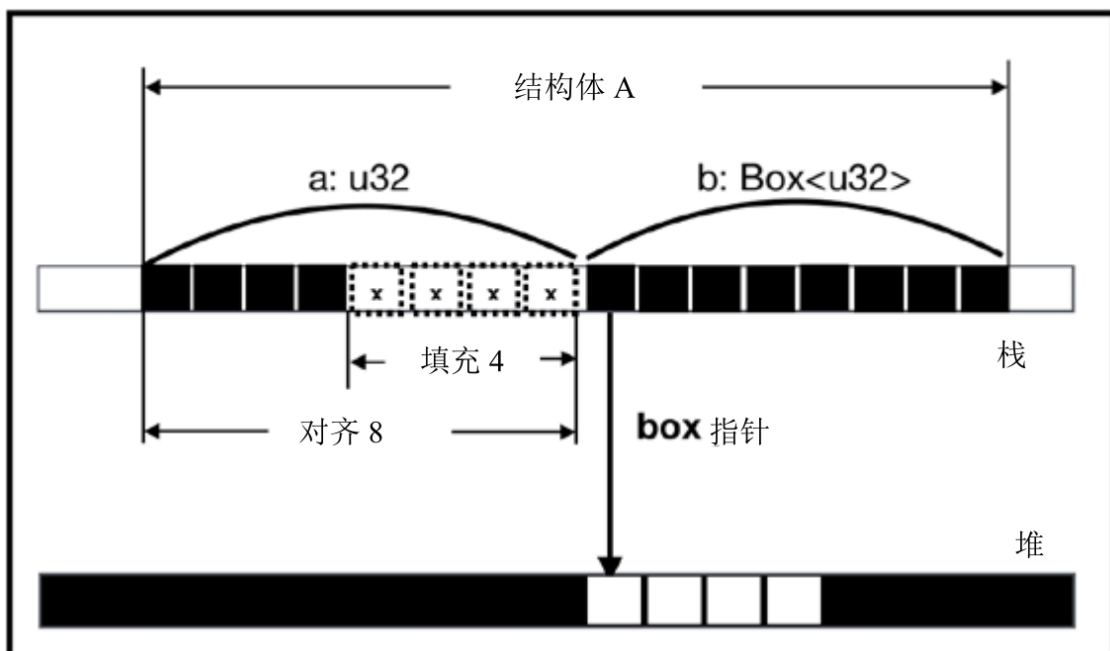


图 4-11 结构体 A 的内存布局

图 4-11 展示了结构体 A 在内存中的布局。结构体 A 包含两个成员：a (类型为 u32) 和 b (类型为 Box<u32>)。成员 a 的起始地址是 0x00000000，长度为 4 字节。成员 b 的起始地址是 0x00000004，长度为 2 字节。成员 b 的值为 0x00000000，指向堆上的 Box 对象。成员 b 的末尾有 4 字节的填充 (padding)。

结构体 A 的总长度为 16 字节。成员 a 的起始地址是 0x00000000，长度为 4 字节。成员 b 的起始地址是 0x00000004，长度为 2 字节。成员 b 的值为 0x00000000，指向堆上的 Box 对象。成员 b 的末尾有 4 字节的填充 (padding)。

图 4-22 展示了结构体 B 的内存布局。结构体 B 包含两个成员：a (类型为 u32) 和 b (类型为 Box<u32>)。成员 a 的起始地址是 0x00000000，长度为 4 字节。成员 b 的起始地址是 0x00000004，长度为 2 字节。成员 b 的值为 0x00000000，指向堆上的 Box 对象。成员 b 的末尾有 4 字节的填充 (padding)。

图 4-22 展示了结构体 B 的内存布局。结构体 B 包含两个成员：a (类型为 u32) 和 b (类型为 Box<u32>)。成员 a 的起始地址是 0x00000000，长度为 4 字节。成员 b 的起始地址是 0x00000004，长度为 2 字节。成员 b 的值为 0x00000000，指向堆上的 Box 对象。成员 b 的末尾有 4 字节的填充 (padding)。

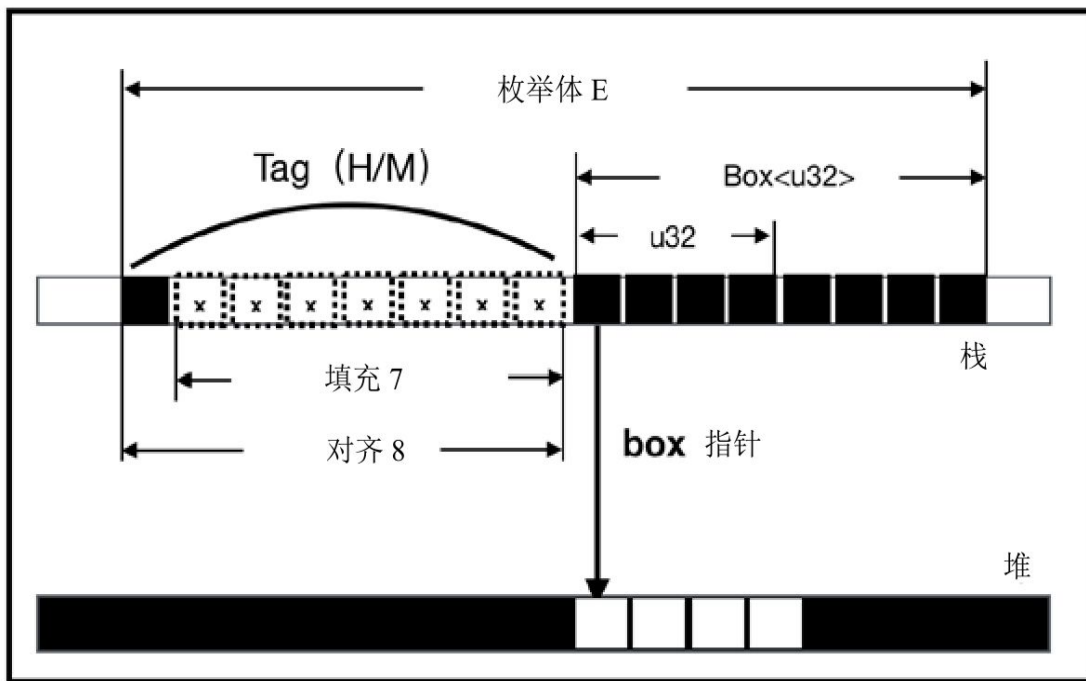


图4-12 枚举体 E 的内存布局

枚举体 E 的内存布局如下：
 H 为枚举体的基类型，M 为枚举体的成员数，Box 为枚举体的基类型。
 H 为枚举体的基类型，M 为枚举体的成员数，Box 为枚举体的基类型。
 H 为枚举体的基类型，M 为枚举体的成员数，Box 为枚举体的基类型。
 H 为枚举体的基类型，M 为枚举体的成员数，Box 为枚举体的基类型。
 H 为枚举体的基类型，M 为枚举体的成员数，Box 为枚举体的基类型。

枚举体 E 的内存布局如下：
 H 为枚举体的基类型，M 为枚举体的成员数，Box 为枚举体的基类型。
 H 为枚举体的基类型，M 为枚举体的成员数，Box 为枚举体的基类型。
 H 为枚举体的基类型，M 为枚举体的成员数，Box 为枚举体的基类型。

图4-22 枚举体 E 的内存布局

图4-23 枚举体 E 的内存布局

Box<u32>: 8

A: 16

B: 16

N: 0

E: 16

U: 8

图4-23 枚举体 E 的内存布局

4.3 枚举

内存管理是C++和GC语言之间的主要区别——内存管理是Rust语言的一个核心特性。

内存管理是Rust语言的一个核心特性。Rust语言通过其内存模型和垃圾回收（GC）来管理内存。

内存管理是Rust语言的一个核心特性。Rust语言通过其内存模型和垃圾回收（GC）来管理内存。RAII（Resource Acquisition Is Initialization）是Rust语言中的一个重要概念，它通过Box和T来管理内存。

RAII（Resource Acquisition Is Initialization）是Rust语言中的一个重要概念。Rc和T是Rust语言中的两个重要概念。RefCell和T是Rust语言中的两个重要概念。Rust语言中的unsafe关键字用于标记不安全代码。

内存管理是Rust语言的一个核心特性。Rust语言通过其内存模型和垃圾回收（GC）来管理内存。

内存管理是Rust语言的一个核心特性。Rust语言通过其内存模型和垃圾回收（GC）来管理内存。5个内存管理模型。

[1] 内存管理是CPU的32Linux/x86内存管理模型

[2] Bertrand Meyer的Eiffel语言中的Design by Contract内存管理模型

5

□ □ □ □ □ □ □ □ □ □

2000年，中国开始实施“西部大开发”战略，旨在促进西部地区的经济发展和基础设施建设。这一战略的实施，使得西部地区的经济活力逐渐增强，基础设施也得到了显著改善。随着西部大开发战略的深入推进，西部地区的经济实力和综合竞争力不断提升，为西部地区的可持续发展奠定了坚实基础。

Rust
 Rust
 C++
 RAII
 Rust

CC++RAIIGCC
5-1

5-1 C++ RAI

```
1. #include <iostream>
2. #include <memory>
3. using namespace std;
4. int main ()
5. {
6.     unique_ptr<int> orig(new int(5));
7.     cout << *orig << endl;
8.     auto stolen = move(orig);
9.     cout << *orig << endl;
10. }
```

5-1 5-2 Rust

5-2 5-1 Rust

```

1. fn main() {
2.     let orig = Box::new(5);
3.     println!("{}", *orig);
4.     let stolen = orig; // Error: use of moved value: `*orig`

5.     println!("{}", *orig);
6. }

```

5-1 unique_ptr Rust Box T 5-1
 5-2 orig orig
 stolen

5-1 C++ move unique_ptr
 stolen orig
 C++ **segmentation fault**

5-2 Rust orig “”
 “” C++ move orig stolen
 Rust

5-2 Rust C++ move
 C++ C++ RAI
 C++
 5-1 Rust Rust
 Rust
 5-2

Rust “”
 “”
 Rust “”
 “”

5.1

Value Reference C++ JavaScript C Ruby Python

GC GC

Vector

Value Semantic Reference Semantic

-
-

Independence

5-1

Figure 5-3: A diagram illustrating the cloning process. It shows a sequence of operations: a box labeled '5-3' followed by a box labeled 'x', then a box labeled 'y', and finally a box labeled 'clone'. Below these boxes, there is a sequence of arrows and boxes: an arrow from 'x' to a box labeled 'x', an arrow from 'y' to a box labeled 'y', and an arrow from 'clone' to a box labeled 'clone'.

```

C++
5-1
Rust Copy
trait

```

5-4 Box T Copy

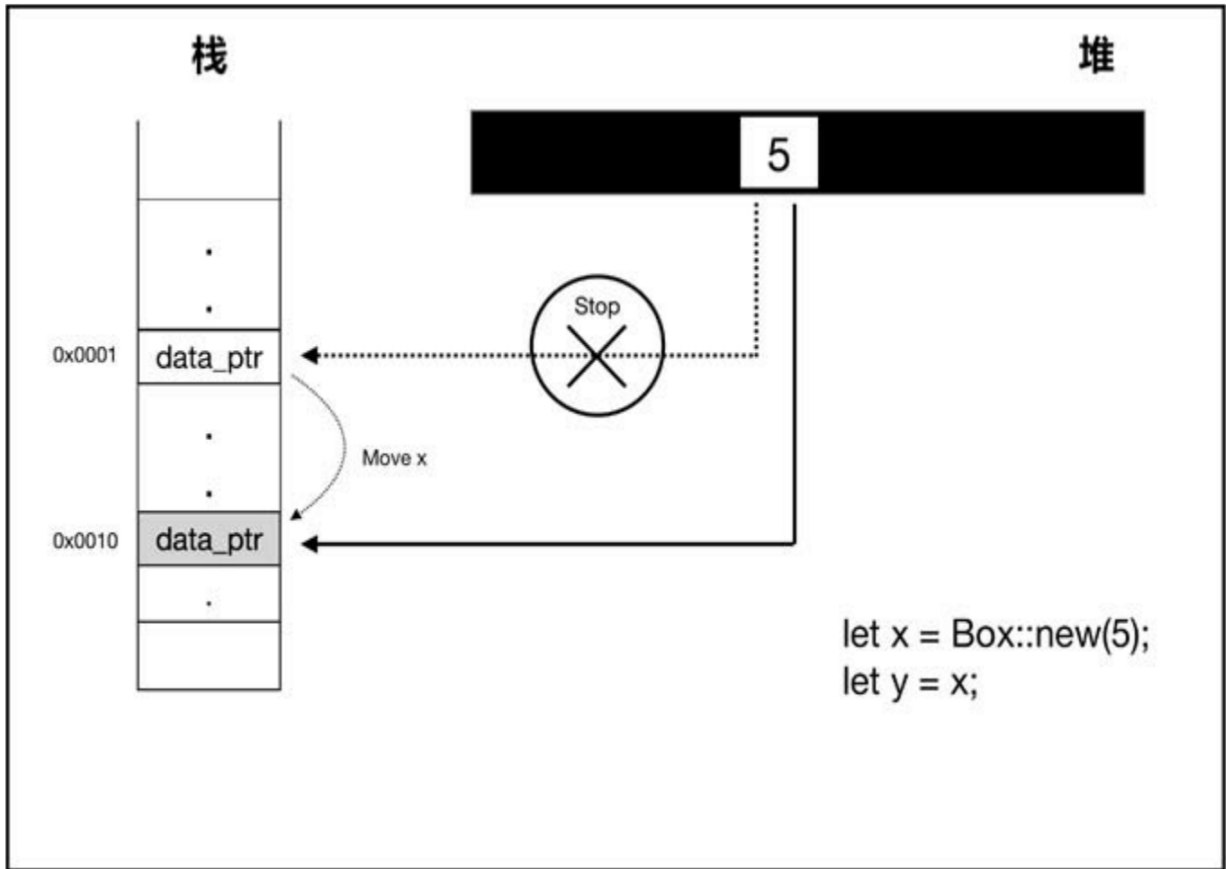
□□□□5-4□□□□□□□□

```
1 | #[derive(Debug, Copy, Clone)]
  |           ^^^^
4 |     b: Box<i32>,
  |     ----- this field does not implement `Copy`
```

Rust Struct A Copy Box T
Copy Clone clone
5.2

Rust Copy trait Rust
Copy Move
5-5 x Box T y
5-2

5-5 Box T
1. fn main() {
2. let x = Box::new(5);
3. let y = x;
4. println!("{:?}", x);
5. }



5-2 Box 与 T 的内存管理

在 Rust 中，`Box` 是一个智能指针，用于在堆上分配内存。它包含一个指向堆内存的指针，并且实现了 `Deref` 和 `DerefMut` trait，使得可以通过 `*` 和 `*` 来访问堆内存。在代码中，`x` 和 `y` 都是 `Box` 类型的变量，它们指向堆内存中的同一块内存（地址 `5-1`）。

在代码中，`x` 和 `y` 都是 `Box` 类型的变量，它们指向堆内存中的同一块内存（地址 `5-5`）。在 Rust 中，`Box` 是一个智能指针，用于在堆上分配内存。它包含一个指向堆内存的指针，并且实现了 `Deref` 和 `DerefMut` trait，使得可以通过 `*` 和 `*` 来访问堆内存。

在 Rust 中，`Box` 是一个智能指针，用于在堆上分配内存。它包含一个指向堆内存的指针，并且实现了 `Deref` 和 `DerefMut` trait，使得可以通过 `*` 和 `*` 来访问堆内存。

Rust 的内存管理模型是基于所有权的。在 Rust 中，`Box` 是一个智能指针，用于在堆上分配内存。它包含一个指向堆内存的指针，并且实现了 `Deref` 和 `DerefMut` trait，使得可以通过 `*` 和 `*` 来访问堆内存。Rust 的内存管理模型是基于所有权的，`Ownership` 是 Rust 的一个核心概念。

在 Rust 中，`Box` 是一个智能指针，用于在堆上分配内存。它包含一个指向堆内存的指针，并且实现了 `Deref` 和 `DerefMut` trait，使得可以通过 `*` 和 `*` 来访问堆内存。

Rust 的内存管理模型是基于所有权的。在 Rust 中，`Box` 是一个智能指针，用于在堆上分配内存。它包含一个指向堆内存的指针，并且实现了 `Deref` 和 `DerefMut` trait，使得可以通过 `*` 和 `*` 来访问堆内存。Rust 的内存管理模型是基于所有权的，`Ownership` 是 Rust 的一个核心概念。Rust 的内存管理模型是基于所有权的，`Ownership` 是 Rust 的一个核心概念。Rust 的内存管理模型是基于所有权的，`Ownership` 是 Rust 的一个核心概念。

· **Linear Logic** “”

· **Affine Logic** 0 1

· Rust

·

Copy 5-1

Copy 5-6

5-6 A

```
1. #[derive(Debug)]
2. struct A{
3.     a: i32,
4.     b: u32,
5. }
6. fn main(){
7.     let a = A {a: 1, b: 2};
8.     let b = a;    // error[E0382]: use of moved value: `a`
9.     println!("{:?}", a);
10. }
```

5-6 8 a println a

A Rust Copy A Copy 5-7

5-7 A Copy

```
1. #[derive(Debug, Copy, Clone)]
2. struct A{
3.     a: i32,
4.     b: u32,
5. }
6. fn main(){
7.     let a = A {a: 1, b: 2};
8.     let b = a;
9.     println!("{:?}", a);
10. }
```

5-7 **[derive Debug Copy Clone]** A Copy A 5-4 Copy

Copy Copy 5-8

5-8 Copy

```
1. fn main(){
2.     let a = ("a".to_string(), "b".to_string());
3.     let b = a;
4.     // println!("{:?}", a);
5.     let c = (1,2,3);
6.     let d = c;
7.     println!("{:?}", c);
8. }
```

5-8 a String a println a c

• 看看“生命周期”章节

Rust 5-10

□□

5-10

```
1. fn main() {
2.     let x = "hello".to_string();
3.     // cannot borrow immutable local variable `x` as mutable
4.     // x += " world";
5. }
```

5-10 4 += x=x+world Rust 3 x

Rust mut Mutable 5-11

5-11 mut

```
1. fn main() {
2.     let mut x = "hello".to_string();
3.     x += " world";
4.     assert_eq!("hello world", x);
5. }
```

5-11 mut x

5.3.2 ——

“”

-
-

lifetime 5-9

a main a main

let 5-12

5-12let

```
1. fn main() {
2.     let a = "hello";
3.     let b = "rust";
4.     let c = "world";
5.     let d = c;
6. }
```

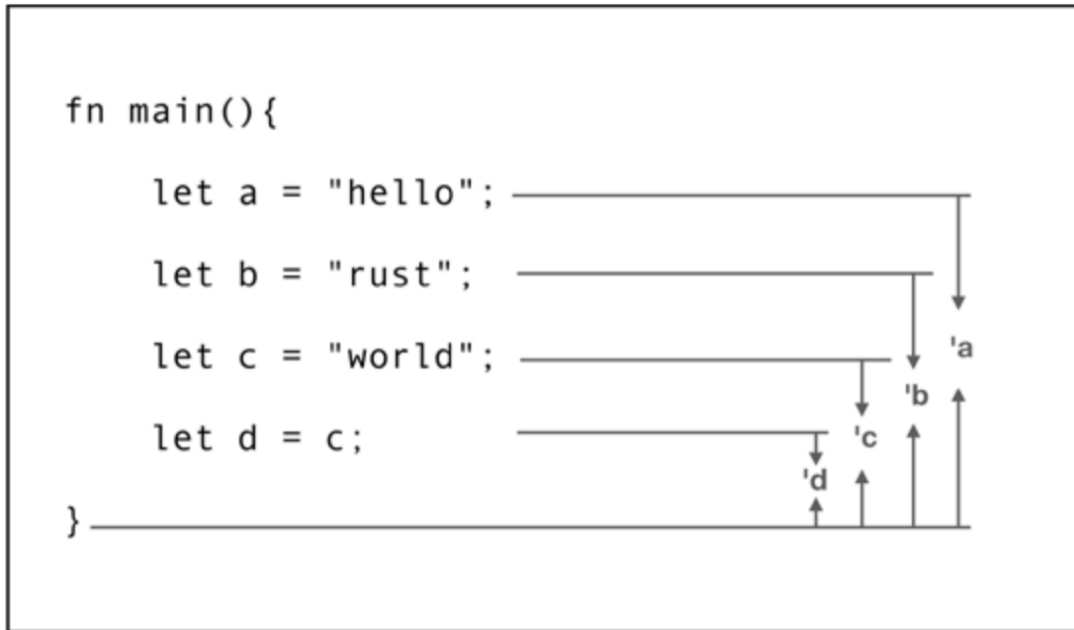
5-124letabcd

```
'a{
    let a = "hello";
    'b {
        let b = "rust";
        'c {
            let c = "world";
            'd {
                let d = c;
            } // 作用域 'd
        } // 作用域 'c
    } // 作用域 'b
} // 作用域 'a
```

abcd

d d c b a
a Rust

5-10c
d c d c
c 5-3



5-3 变量生命周期

变量生命周期是指变量从创建到销毁的时间段。在 Rust 中，变量的生命周期是由编译器和运行时共同决定的。变量的生命周期可以分为两个部分：编译时生命周期和运行时生命周期。

编译时生命周期是指变量在编译时被创建和销毁的时间段。

运行时

生命周期是指变量在运行时被创建和销毁的时间段。

在 Rust 中，变量的生命周期是由编译器和运行时共同决定的。

```

1. fn main(){
2.     let outer_val = 1;
3.     let outer_sp = "hello".to_string();
4.     {
5.         let inner_val = 2;
6.         outer_val;
7.         outer_sp;
8.     }
9.     println!("{:?}", outer_val);
10.    // error[E0425]: cannot find value `inner_val` in this scope
11.    // println!("{:?}", inner_val);
12.    // error[E0382]: use of moved value: `outer_sp`
13.    // println!("{:?}", outer_sp);
14. }

```

5-13 4 8 main 5
 let inner_val
 11 10

2 outer_val
 9 outer_val 3 outer_sp
 outer_sp
 13 12

match
 match

match

match 5-14

5-14 match

```

1. fn main(){
2.     let a = Some("hello".to_string());
3.     match a { // -----
4.         Some(s) => println!("{:?}", s),      // | match scope
5.         _ => println!("nothing")             // |
6.     } // -----
7.     // error[E0382]: use of partially moved value: `a`
8.     // println!("{:?}", a);
9. }

```

5-14 match 使用 `String` 和 `Option` 类型

```

match a {
    Some(s) => println!("{}", s),
    _ => println!("nothing")
}

```

8 行代码，7 行代码，match 使用 `String` 和 `Option` 类型

4 行代码

for loop 和 while 循环

5-15 for 循环

```

1. fn main(){
2.     let v = vec![1,2,3];
3.     for i in v { // -----
4.         println!("{:?}", i);      // |
5.         // error[E0382]: use of moved value: `v`      // | for scope
6.         // println!("{:?}", v);    // |
7.     } //-----
8. }

```

5-15 使用 `vec` 和 `for` 循环

if let 和 while let

if let 和 while let 循环

5-16 if let 循环

```

1. fn main() {
2.     let a = Some("hello".to_string());
3.     if let Some(s) = a { // -----
4.         println!("{:?}", s) //| if let scope
5.     } //-----
6. }

```

5-16 a Option String 3 if
let a if let

5-17 while let

5-17 while let

```

1. fn main() {
2.     let mut optional = Some(0);
3.     while let Some(i) = optional {
4.         if i > 9 {
5.             println!("Greater than 9, quit!");
6.             optional = None;
7.         } else {
8.             println!("`i` is `{:?}`. Try again.", i);
9.             optional = Some(i + 1);
10.        }
11.    }
12. }

```

5-17 optional Option i32 i32
Option i32 while let i

5-18

5-18

```

1. fn foo(s: String) -> String { // ----
2.     let w = " world".to_string(); // | function scope
3.     s + &w                        // |
4. } // -----
5. fn main() {
6.     let s = "hello".to_string();
7.     let ss = foo(s);
8.     // println!("{:?}", s) // Error: use of moved value: `s`
9. }

```

5-18 `s: String` を `foo` に渡す
`main` で `s` を使おうとすると

エラー

原因は、`s` が `foo` に渡された後、`s` のメモリが解放されるから

・ `s` を `&T` のように参照渡しで渡す

・ `move` を使って、`s` を移動させる

・ `&mut` を使って、`s` を可変参照にする

5-19 `s` を `String` の代わりに `StringView` に変更する

5-19 `s` を `StringView` の代わりに `String` に変更する

```

1. fn main() {
2.     let s = "hello".to_string();
3.     let join = |i: &str| {s + i}; // moved s into closure scope
4.     assert_eq!("hello world", join(" world"));
5.     // println!("{:?}", s); // error[E0382]: use of moved value: `s`
6. }

```

5-19 `s: String` を `StringView` に変更する
`main` で `s` を使おうとすると

5.4 参照渡し

5-20 `s` を `StringView` の代わりに `String` に変更する

5-20

```
1. fn foo(mut v: [i32; 3]) -> [i32; 3] {
2.     v[0] = 3;
3.     assert_eq!([3,2,3], v);
4.     v
5. }
6. fn main() {
7.     let v = [1,2,3];
8.     foo(v);
9.     assert_eq!([1,2,3], v);
10. }
```

5-20 [T]N foo v [i32]3 3 i32 mut foo 0

main 7 v foo v foo v[3 2 3] main v[1 2 3] 9

7 v foo let v

foo Rust 5-21

5-21

```
1. fn foo(v: &mut [i32; 3]) {
2.     v[0] = 3;
3. }
4. fn main() {
5.     let mut v = [1,2,3];
6.     foo(&mut v);
7.     assert_eq!([3,2,3], v);
8. }
```

5-21 5-20 &mut v foo
foo v
6 &mut v foo 7
v

Reference Rust
Alias
& &mut

&x x **Borrowing** &
owner

-
-

5-22

5-22

```

1. fn bubble_sort(a: &mut Vec<i32>) {
2.     let mut n = a.len(); // 获取数组长度
3.     while n > 0 {
4.         // 初始化遍历游标, max_ptr 始终指向最大值
5.         let (mut i, mut max_ptr) = (1, 0);
6.         // 冒泡开始, 如果前者大于后者, 则互换位置, 并设置当前最大值游标
7.         while i < n {
8.             if a[i-1] > a[i] {
9.                 a.swap(i-1, i);
10.                max_ptr = i;
11.            }
12.            i += 1;
13.        }
14.        // 本次遍历的最大值位置也是下一轮冒泡的终点
15.        n = max_ptr;
16.    }
17. }
18. fn main() {
19.     let mut a = vec![1, 4, 5, 3, 2];
20.     bubble_sort(&mut a);
21.     println!("{:?}", a); // [1, 2, 3, 4, 5]
22. }

```

5-22 bubble_sort 2 a.len
 a.len
 9 a.swap i-1 i

5-23 std::vec::Vec::len

5-23 len

```

1. pub fn len(&self) -> usize {
2.     self.len
3. }

```

5-23 len &self self &self
 self len

内存模型是操作系统和硬件共同决定的，与具体的编程语言无关。

内存模型

内存模型是操作系统和硬件共同决定的，与具体的编程语言无关。

- 内存模型是操作系统和硬件共同决定的，与具体的编程语言无关。
- 内存模型是操作系统和硬件共同决定的，与具体的编程语言无关。Alias
- 内存模型是操作系统和硬件共同决定的，与具体的编程语言无关。

内存模型是操作系统和硬件共同决定的，与具体的编程语言无关。

Rust内存模型是操作系统和硬件共同决定的，与具体的编程语言无关。

内存模型是操作系统和硬件共同决定的，与具体的编程语言无关。Rust

内存模型

内存模型是操作系统和硬件共同决定的，与具体的编程语言无关。

内存模型是操作系统和硬件共同决定的，与具体的编程语言无关。5-4

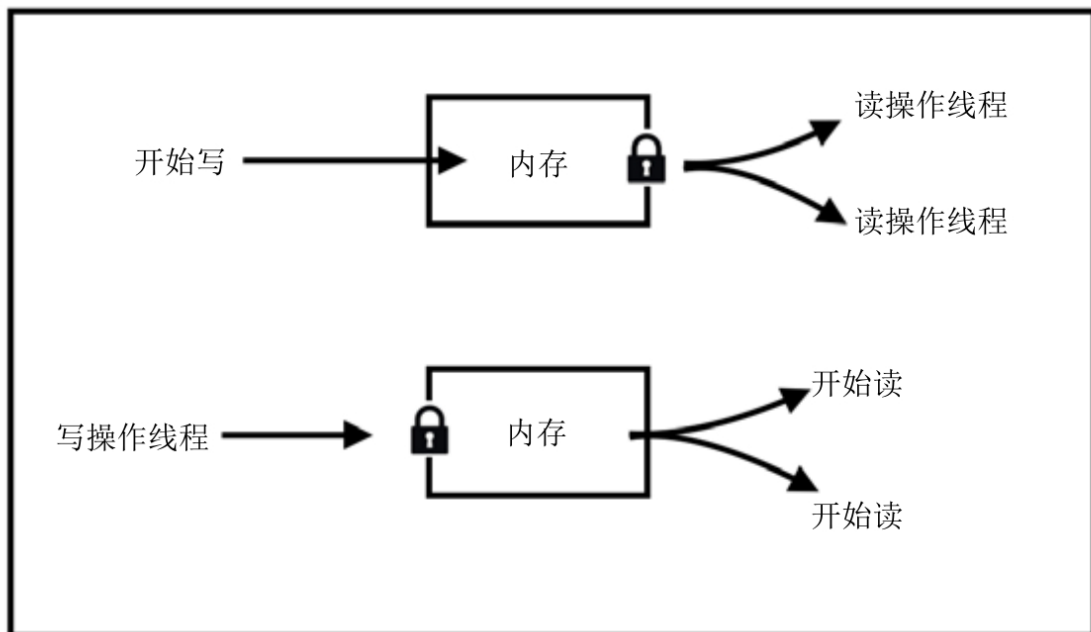


图5-4 内存模型

内存模型是操作系统和硬件共同决定的，与具体的编程语言无关。

内存模型是操作系统和硬件共同决定的，与具体的编程语言无关。5-24

内存模型是操作系统和硬件共同决定的，与具体的编程语言无关。5-24

```

1. fn compute(input: &u32, output: &mut u32) {
2.     if *input > 10 {
3.         *output = 1;
4.     }
5.     if *input > 5 {
6.         *output *= 2;
7.     }
8. }
9. fn main() {
10.    let i = 20;
11.    let mut o = 5;
12.    compute(&i, &mut o); // o = 2
13. }

```

5-24 Rust 的 `compute` 函数中，`output` 参数是 `&mut u32`，表示它是一个可变的引用。

5-25 Rust 的 `compute` 函数中，`input` 参数是 `&u32`，表示它是一个不可变的引用。

5-25 Rust 的 `compute` 函数中，`input` 参数是 `&u32`，表示它是一个不可变的引用。

```

1. fn main() {
2.     let mut i = 20;
3.     compute(&i, &mut i);
4. }

```

5-25 Rust 的 `compute` 函数中，`input` 参数是 `&u32`，表示它是一个不可变的引用。在 `main` 函数中，`i` 是一个 `mut` 变量，因此 `&i` 是一个可变的引用。但是，`compute` 函数要求 `input` 是一个不可变的引用，这会导致编译错误。在 `compute` 函数中，`input` 参数是 `&u32`，表示它是一个不可变的引用。在 `main` 函数中，`i` 是一个 `mut` 变量，因此 `&i` 是一个可变的引用。但是，`compute` 函数要求 `input` 是一个不可变的引用，这会导致编译错误。

5-25 Rust 的 `compute` 函数中，`input` 参数是 `&u32`，表示它是一个不可变的引用。

error[E0502]: cannot borrow `i` as mutable because it is also borrowed as immutable

compute Rust
input output compute
5-26

5-26 compute

```
1. fn compute(input: &u32, output: &mut u32) {
2.     let cached_input = *input;
3.     if cached_input > 10 {
4.         *output = 2;
5.     } else if cached_input > 5 {
6.         *output *= 2;
7.     }
8. }
9. fn main() {
10.    let i = 20;
11.    let mut o = 5;
12.    compute(&i, &mut o); // o = 2
13. }
```

5-26 Rust
input output 2 cached_input
input cached_input if if else
10 5

Rust

-
-
-
-

&String 5-27

5-27 &String

```

1. fn join(s: &String) -> String {
2.     let append = *s;
3.     "Hello".to_string() + &append
4. }
5. fn main(){
6.     let x = " hello".to_string();
7.     join(&x);
8. }

```

5-27 join &String 2 let

```

error[E0507]: cannot move out of borrowed content
|     let append = *s;
|                   ^^

```

s append s
 main x &x Rust

5.5

Rust borrow checker
 Rust
 5-28

5-28

```

1. fn main() {
2.     let r; // 'a -----
3.     {                                           //
4.         let x = 5; // 'b -----
5.         r = &x;    //
6.     } // -----
7.     println!("r: {}", r); //
8. } // -----

```

5-28 `r` `main` `a` `x`
 3 6 `b` `a` 5
 Rust 5-28

```

error[E0597]: `x` does not live long enough
5 |         r = &x;
  |             - borrow occurs here
6 |     }
  |       ^ `x` dropped here while still borrowed
7 |     println!("r: {}", r);
8 | }
  | - borrowed value needs to live until here

```

5-28 `&x`
`r` `a` `x` `b` `a`
`b`

Rust

5.5.1

`a` `&`

return_str
s &s[..]
Rust
String

5-30

5-30foo

```
1. fn foo<'a>(x: &'a str, y: &'a str) -> &'a str {  
2.     let result = String::from("really long string");  
3.     // error[E0597]: `result` does not live long enough  
4.     result.as_str()  
5. }  
6. fn main() {  
7.     let x = "hello";  
8.     let y = "rust";  
9.     foo(x, y);  
10. }
```

5-303foo

5-31

5-31

```
1. fn the_longest(s1: &str, s2: &str) -> &str {  
2.     if s1.len() > s2.len() { s1 } else { s2 }  
3. }  
4. fn main() {  
5.     let s1 = String::from("Rust");  
6.     let s1_r = &s1;  
7.     {  
8.         let s2 = String::from("C");  
9.         let res = the_longest(s1_r, &s2);  
10.        println!("{}", res);  
11.    }  
12. }
```

5-31

```
error[E0106]: missing lifetime specifier
| fn the_longest(s1: &str, s2: &str) -> &str {
|                                     ^ expected lifetime parameter
```

5-32

5-32 the_longest

```
1. fn the_longest<'a>(s1: &'a str, s2: &'a str) -> &'a str {
2.     if s1.len() > s2.len() { s1 } else { s2}
3. }
4. fn main() {
5.     let s1 = String::from("Rust");
6.     let s1_r = &s1;
7.     {
8.         let s2 = String::from("C");
9.         let res = the_longest(s1_r, &s2);
10.        println!("{}", res); // Rust is the longest
11.    }
12. }
```

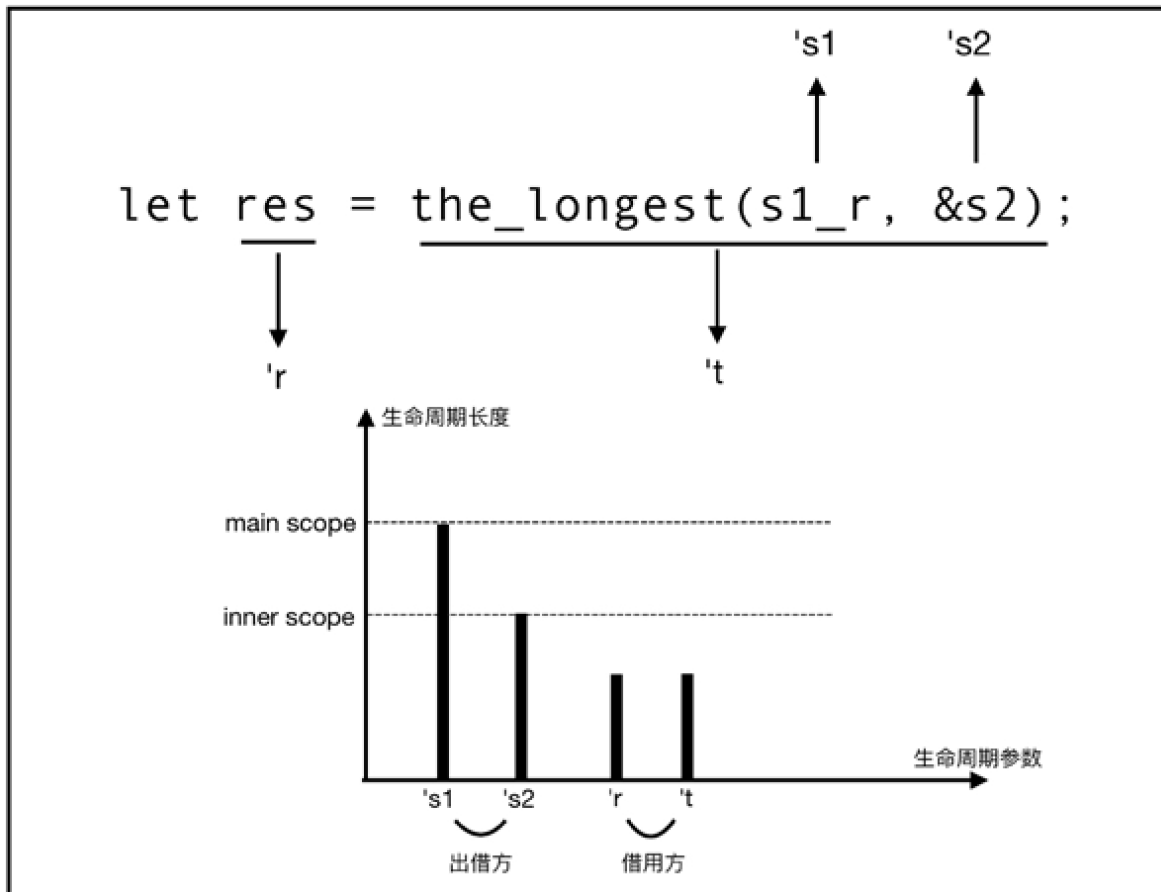
5-32 the_longest

&a str

a

main the_longest

5-5



5-5

the_longest s1_r 5-32 6
12 s1 the_longest a
s1 &s2 5-32 7 10
s2 a s2

the_longest s1_r &s2 t 9 let
res r res the_longest
res r t

5-33

5-33

1. fn the_longest<'a, 'b>(s1: &'a str, s2: &'b str) -> &'a str {
2. if s1.len() > s2.len() { s1 } else { s2 }
3. }

```
error[E0312]: lifetime of reference outlives lifetime of borrowed
content...
```

5-34

```

000005-34000b00a 0000000000000000b 000000000000000000
00b outlive0a 0000000000000000b 000a 00b 00000000
00000main 000000000 s1_r 000000000&s2 0000000000
_longest00000000s100000000s2000s100000000a 0s20000000
00000000000000a 000000000b 000000000b00a 000b 00000000
. 000

```

```
the_longest[a] b  
[s1][s2][a] t res  
r t Rust let  
res &s2 s1_r res &s2 s1_r res  
&s2 Rust res r  
&s2 s2 s2 r s2  
r s2 r b a b a  
s1_r s1 a s1 r  
s1 r a a 5-6
```



```

1. struct Foo<'a> {
2.     part: &'a str,
3. }
4. fn main() {
5.     let words = String::from("Sometimes think, the greatest sorrow than
older");
6.     let first = words.split(',').next().expect("Could not find a
', '");
7.     let f = Foo { part: first };
8.     assert_eq!("Sometimes think", f.part);
9. }

```

5-35 Foo &str a
part &a str

main String words split
words next expect first next
6

7 first Foo
part part main Foo f
part f first first f.part
Rust

Foo 5-36
impl Foo

5-36 Foo

1.#[derive(Debug)]

```

2. struct Foo<'a> {
3.     part: &'a str,
4. }
5. impl<'a> Foo<'a> {
6.     fn split_first(s: &'a str) -> &'a str {
7.         s.split(',').next().expect("Could not find a ', '")
8.     }
9.     fn new(s: &'a str) -> Self {
10.        Foo {part: Foo::split_first(s)}
11.    }
12. }
13. fn main() {
14.    let words = String::from(
15.        "Sometimes think, the greatest sorrow than older");
16.    println!("{}", Foo::new(words.as_str()));
17. }

```

```

Foo::new::first::first::new
::a ::impl::impl
::new::first

```

error[E0495]: cannot infer an appropriate lifetime for lifetime parameter in function call due to conflicting requirements

```

::a
Foo

```

```

Rust static static
static &static str
5-37

```

5-37

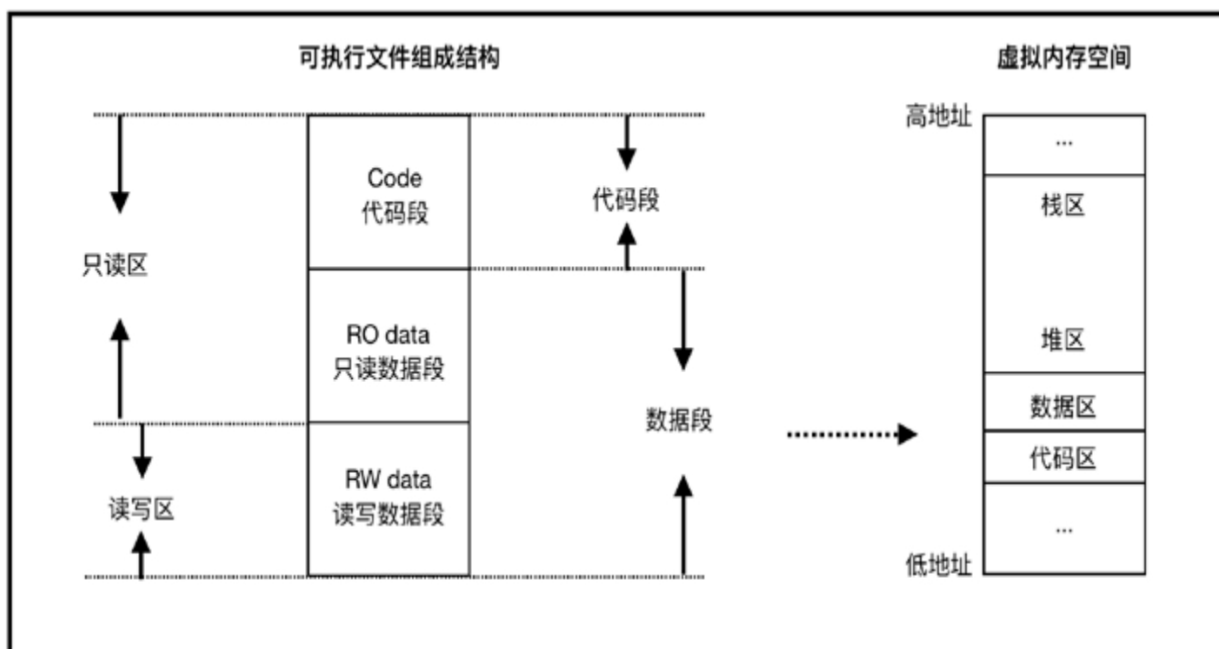
```

1. fn main() {
2.     let x = "hello Rust";
3.     let y = x;
4.     assert_eq!(x, y);
5. }

```

5-37 `x` `y=x` `x`

5-7



5-7

5-37 `x`

Rust 2018 `const` `static` `static`

5.5.2

5-38

5-38

```

1. fn first_word(s: &str) -> &str {
2.     let bytes = s.as_bytes();
3.     for (i, &item) in bytes.iter().enumerate() {
4.         if item == b' ' {
5.             return &s[0..i];
6.         }
7.     }
8.     &s[..]
9. }
10. fn main() {
11.     println!("{:?}", first_word("hello Rust"));
12. }

```

5-38 first_word &str as_bytes for &s[0..i] 3 for enumerate for &item item 4

Rust Rust

Lifetime Elision Rule

-
-
- &self &mut self self

5-39

5-39

```

1.  fn print(s: &str);                                // 省略
2.  fn print<'a>(s: &'a str);                          // 展开
3.  fn debug(lvl: uint, s: &str);                      // 省略
4.  fn debug<'a>(lvl: uint, s: &'a str);               // 展开
5.  fn substr(s: &str, until: uint) -> &str;          // 省略
6.  fn substr<'a>(s: &'a str, until: uint) -> &'a str; // 展开
7.  fn get_str() -> &str;                              // 非法
8.  fn frob(s: &str, t: &str) -> &str;                // 非法
9.  fn get_mut(&mut self) -> &mut T;                  // 省略
10. fn get_mut<'a>(&'a mut self) -> &'a mut T;        // 展开
11. // 省略
12. fn args<T:ToCStr>(&mut self, args: &[T]) -> &mut Command
13. // 展开
14. fn args<'a, 'b, T:ToCStr>(&'a mut self, args: &'b [T]) -> &'a mut
    Command
15. fn new(buf: &mut [u8]) -> BufWriter;              // 省略
16. fn new<'a>(buf: &'a mut [u8]) -> BufWriter<'a>    // 展开

```

5-39

1

[illegible]

8

[illegible]

```

    12  &mut self
    self

```

5-36 5-40

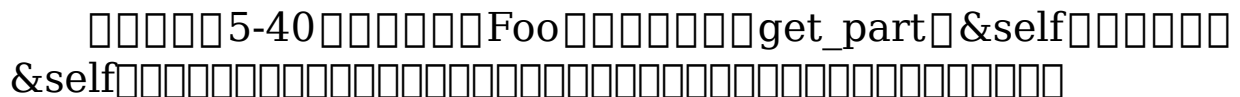
11

5-40 5-36 get_part


```

1. #[derive(Debug)]
2. struct Foo<'a> {
3.     part: &'a str,
4. }
5. impl<'a> Foo<'a> {
6.     fn split_first(s: &'a str) -> &'a str {
7.         s.split(',').next().expect("Could not find a ', '")
8.     }
9.     fn new(s: &'a str) -> Self {
10.        Foo {part: Foo::split_first(s)}
11.    }
12.    fn get_part(&self) -> &str {
13.        self.part
14.    }
15. }
16. fn main() {
17.    let words = String::from("Sometimes think, the greatest sorrow than
    older");
18.    let foo = Foo::new(words.as_str());
19.    println!("{:?}", foo.get_part());
20. }

```



5.5.3 特性











```

1. use std::fmt::Debug;
2. #[derive(Debug)]
3. struct Ref<'a, T: 'a>(&'a T);
4. fn print<T>(t: T)
5. where
6.     T: Debug,
7. {
8.     println!("`print`: t is {:?}" , t);
9. }
10. fn print_ref<'a, T>(t: &'a T)
11. where
12.     T: Debug + 'a,
13. {
14.     println!("`print_ref`: t is {:?}" , t);
15. }
16. fn main() {
17.     let x = 7;
18.     let ref_x = Ref(&x);
19.     print_ref(&ref_x);
20.     print(ref_x);
21. }

```

5-41 Ref T
 T T a T a
 Ref a print print_ref
 print_ref

&T
 T static
 static static Hardcode

Rust 2018 5-41 T a

5.5.4 trait

trait trait 5-42
5-42 trait trait

```
1. trait Foo {}
2. struct Bar<'a> {
3.     x: &'a i32,
4. }
5. impl<'a> Foo for Bar<'a> {}
6. fn main() {
7.     let num = 5;
8.     let box_bar = Box::new(Bar { x: &num });
9.     let obj = box_bar as Box<Foo>;
10. }
```

5-42 Bar trait Foo 8 Box new Bar 9 trait Box Foo

trait

- trait static
- trait &a X &a mut X a
- trait T a a
- trait T a 5-43

5-43 trait

```
1. trait Foo<'a> {}
2. struct FooImpl<'a> {
3.     s: &'a [u32],
4. }
5. impl<'a> Foo<'a> for FooImpl<'a> {
6. }
7. fn foo<'a>(s: &'a [u32]) -> Box<Foo<'a>> {
8.     Box::new(FooImpl { s: s })
9. }
10. fn main() {}
```


String

Box<T> deref 5-46 Box<T> deref

5-46 Box<T> deref

```
1. impl<T: ?Sized> Deref for Box<T> {
2.     type Target = T;
3.     fn deref(&self) -> &T {
4.         &**self
5.     }
6. }
```

deref &T 5-45 a b &T *a *b *a.deref() *b.deref() Box<T> T 5-45 b

Box<T> * Rust trait DerefMove Rust trait Box<T>

5-47 Rc<T> Arc<T>

5-47 Rc<T> Arc<T>

```
1. use std::rc::Rc;
2. use std::sync::Arc;
3. fn main(){
4.     let r = Rc::new("Rust".to_string());
5.     let a = Arc::new(vec![1.0, 2.0, 3.0]);
6.     // error[E0507]: cannot move out of borrowed content
7.     // let x = *r;
8.     // println!("{:?}", r);
9.     // error[E0507]: cannot move out of borrowed content
10.    // let f = *foo;
11. }
```

Box<T> owned_box<T> 5-48 Box<T>

5-48 Box<T>

1. #[lang = "owned_box"]
2. pub struct Box<T: ?Sized> (Unique<T>);

Box<T> Lang Item owned_box Box<T>
Box<T> bool
Lang Item

Box<T> box 5-49
Box new Rc new Arc new

5-49 Box new Rc new Arc new

1. impl<T> Box<T> {
2. pub fn new(x: T) -> Box<T> {
3. box x
4. }
5. }
6. impl<T> Rc<T> {
7. pub fn new(value: T) -> Rc<T> {
8. unsafe {
9. Rc {
10. ptr: Shared::new(Box::into_raw(box RcBox {})),
11. }
12. }
13. }
14. }
15. impl<T> Arc<T> {
16. pub fn new(data: T) -> Arc<T> {
17. let x: Box<_> = box ArcInner {};
18. }
19. }

5-49
box box Rust API

boxexchange_mallocbox_free
5-50

5-50box_freeexchange_malloc

```
1. #[cfg_attr(not(test), lang = "box_free")]
2. #[inline]
3. pub(crate) unsafe fn box_free<T: ?Sized>(ptr: *mut T) {
4.     ...
5.     Heap.dealloc(ptr as *mut u8, layout);
6. }
7. #[cfg(not(test))]
8. #[lang = "exchange_malloc"]
9. #[inline]
10. unsafe fn exchange_malloc(size: usize, align: usize) -> *mut u8 {
11.     ...
12.     Heap.alloc(layout)...
13. }
```

5-50box_freeexchange_malloc
Lang Itembox_freedealloc
exchange_mallocalloc

5.6.1 RcTWeakT

reference countingGC
RustRcTGC Rust

RustRcT
RcT
RcTRust
5-51

5-51RcT

```

1. use std::rc::Rc;
2. fn main() {
3.     let x = Rc::new(45);
4.     let y1 = x.clone(); // 增加强引用计数
5.     let y2 = x.clone(); // 增加强引用计数
6.     println!("{:?}", Rc::strong_count(&x));
7.     let w = Rc::downgrade(&x); // 增加弱引用计数
8.     println!("{:?}", Rc::weak_count(&x));
9.     let y3 = &*x;           // 不增加计数
10.    println!("{}", 100 - *x);
11. }

```

Rc<T> 使用 std::rc::use 创建 Rc 3 个 x 4 5 clone 6 Rc::strong_count 3 clone

clone 7 downgrade Weak<T> Rc<T> Weak<T> Rc::strong_count Rc::weak_count

4 Rc<T> " " Weak<T> Weak<T> 5-52

5-52 Weak<T>


```

1. use std::rc::Rc;
2. use std::rc::Weak;
3. use std::cell::RefCell;
4. struct Node {
5.     next: Option<Rc<RefCell<Node>>>,
6.     head: Option<Weak<RefCell<Node>>>
7. }
8. impl Drop for Node {
9.     fn drop(&mut self) {
10.         println!("Dropping!");
11.     }
12. }
13. fn main() {
14.     let first = Rc::new(RefCell::new(Node { next: None, head: None }));
15.     let second = Rc::new(RefCell::new(Node { next: None, head:
None }));
16.     let third = Rc::new(RefCell::new(Node { next: None, head: None }));
17.     first.borrow_mut().next = Some(second.clone());
18.     second.borrow_mut().next = Some(third.clone());
19.     third.borrow_mut().head = Some(Rc::downgrade(&first));
20. }

```

5-52 Node head Node next Node

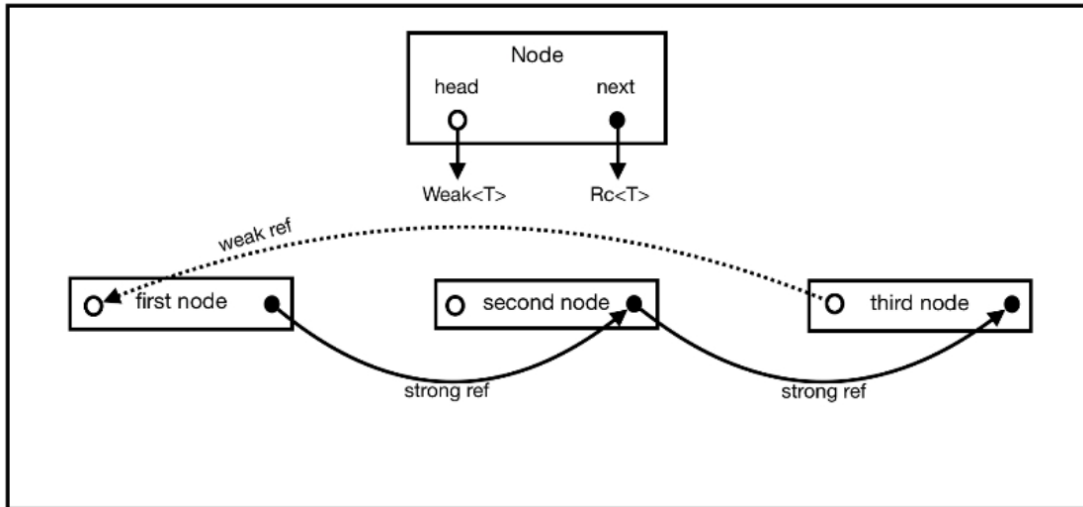
14 16 next head None

17 second first next first second

18 third second next second third

19 third first head third first

6-8



5-8 Weak<T> 内存管理

5-52 内存管理 drop 内存管理 Arena 内存管理

5.6.2 Cell<T> RefCell<T>

Rust 内存管理 Cell<T> RefCell<T> Interior Mutability

Cell<T>

Rust Struct 5-53 Cell<T>

5-53 Cell<T>

```

1. use std::cell::Cell;
2. struct Foo {
3.     x: u32,
4.     y: Cell<u32>
5. }
6. fn main(){
7.     let foo = Foo { x: 1 , y: Cell::new(3)};
8.     assert_eq!(1, foo.x);
9.     assert_eq!(3,foo.y.get());
10.    foo.y.set(5);
11.    assert_eq!(5,foo.y.get());
12. }

```

5-53 Foo x u32 y Cell u32
main foo 9 get Cell
u32 10 set y

Cell T set/get OOP
setter/getter Cell T set/get
Cell T T

Cell T Copy T get
get T set
Cell T Rust Copy T
Copy T get_mut Rust

Cell T 5-49
Cell T get/set

RefCell T

Copy Cell T Rust RefCell T
T Copy 5-54

5-54 RefCell T

```

1. use std::cell::RefCell;
2. fn main() {
3.     let x = RefCell::new(vec![1,2,3,4]);
4.     println!("{:?}", x.borrow());
5.     x.borrow_mut().push(5);
6.     println!("{:?}", x.borrow());
7. }

```

RefCell<T>のborrow/borrow_mutはCell<T>のget/set
 RefCell<T>の内部状態を参照する際に、 borrow/borrow_mut
 を呼び出す際に、 borrow/borrow_mutが呼び出されている間は、
 panic!を返す5-55

5-55 RefCell<T>のpanic

```

1. use std::cell::RefCell;
2. fn main() {
3.     let x = RefCell::new(vec![1,2,3,4]);
4.     let mut mut_v = x.borrow_mut();
5.     mut_v.push(5);
6.     // thread 'main' panicked at 'already borrowed: BorrowMutError',
7.     // let mut mut_v2 = x.borrow_mut();
8. }

```

5-55の7のborrow_mutは、 Rustのmain関数の
 6のpanic!を返す

Cell<T>のRefCell<T>の内部状態を参照する際に、
 5-49のRc<RefCell<T>>のCell<T>のRefCell<T>の
 内部状態を参照する

- Cell<T>のset/getは、 RefCell<T>のborrow/borrow_mutは、 Ref<T>のRefMut<T>の内部状態を参照する

- Cell<T>のCopyは、 RefCell<T>のCopyは、 Copy

- Cell<T>のpanicは、 RefCell<T>のpanicは、 panic

CellT RefCellT
CellT RefCellT

5.6.3 CowT

Copy on WriteLinux
“” Rust
CowT

CowT

- Borrowed
- Owned

OptionTOptionT“”“” CowT
“”“”

CowT
CowTDerefCowT
5-56

5-56CowT

```
1. use std::borrow::Cow;
2. fn abs_all(input: &mut Cow<i32>) {
3.     for i in 0..input.len() {
4.         let v = input[i];
5.         if v < 0 {
6.             input.to_mut()[i] = -v;
7.         }
8.     }
9. }
10. fn abs_sum(ns: &i32) -> i32 {
11.     let mut lst = Cow::from(ns);
12.     abs_all(&mut lst);
13.     lst.iter().fold(0, |acc, &n| acc + n)
14. }
15. fn main() {
16.     // 这里没有可变需求, 所以不会克隆
17.     let s1 = [1,2,3];
18.     let mut i1 = Cow::from(&s1[..]);
19.     abs_all(&mut i1);
20.     println!("IN: {:?}", s1);
21.     println!("OUT: {:?}", i1);
22.     // 这里有可变需求, 所以会克隆
```

```

23.      // 注意：借用数据被克隆为了新的对象
24.      //s2 != i2. 实际上，s2 不可变，也不会被改变
25.      let s2 = [1,2,3, -45, 5];
26.      let mut i2 = Cow::from(&s2[..]);
27.      abs_all(&mut i2);
28.      println!("IN: {:?}", s2);
29.      println!("OUT: {:?}", i2);
30.      //这里不会克隆，因为数据本身拥有所有权
31.      //注意：在本例中，v1 本身就是可变的
32.      let mut v1 = Cow::from(vec![1,2,-3,4]);
33.      abs_all(&mut v1);
34.      println!("IN/OUT: {:?}", v1);
35.      //没有可变需求，所以没有克隆
36.      let s3 = [1,3,5,6];
37.      let sum1 = abs_sum(&s3[..]);
38.      println!("{:?}", s3);
39.      println!("{}", sum1);
40.      //这里有可变需求，所以发生了克隆
41.      let s4 = [1,-3,5,-6];
42.      let sum2 = abs_sum(&s4[..]);
43.      println!("{:?}", s4);
44.      println!("{}", sum2);
45.  }

```

5-56 abs_all Cow
 [i32] 0 to_mut
 abs_sum abs_all

17 21 s1 i1 i1 Cow from
 &mut i1 abs_all s1 0 if
 to_mut 20 21

IN: [1, 2, 3]

OUT: [1, 2, 3]

25 29 s2 0 abs_all if
 to_mut to_mut for

2829

IN: [1, 2, 3, -45, 5]

OUT: [1, 2, 3, 45, 5]

3234v1CowTVecT
abs_allifto_mut

IN/OUT: [1, 2, 3, 4]

3639abs_sumabs_all
s3to_mut

[1, 3, 5, 6]

15

4144s4to_mut

[1, -3, 5, -6]

15

5-56CowT

· CowTDerefT

· T to_mut
Tto_mut

· Tinto_ownedT

CowT Token
tokentoken&str
StringCowT5-57

5-57CowT


```

1. use std::borrow::Cow;
2. use std::thread;
3. #[derive(Debug)]
4. struct Token<'a> {
5.     raw: Cow<'a, str>,
6. }
7. impl<'a> Token<'a> {
8.     pub fn new<S>(raw: S) -> Token<'a>
9.     where
10.         S: Into<Cow<'a, str>>,
11.     {
12.         Token { raw: raw.into() }
13.     }
14. }
15. fn main() {
16.     let token = Token::new("abc123");
17.     let token = Token::new("api.example.io".to_string());
18.     thread::spawn(move || {
19.         println!("token: {:?}", token);
20.     }).join().unwrap();
21. }

```

```

5-57 Token Cow a str main
& static str String into Cow
a str String

```

```

token
main 5-58

```

5-58

```

1. fn main() {
2.     let raw = String::from("abc");
3.     let s = &raw[..];
4.     let token = Token::new(s);
5.     thread::spawn(move || {
6.         println!("token: {:?}", token);
7.     }).join().unwrap();
8. }

```

5-58

error[E0597]: `raw` does not live long enough

5.7

Rust trait `Send` and `Sync` are used to ensure that data is safe to be shared across threads. Rust trait `Send` and `Sync` are used to ensure that data is safe to be shared across threads.

- `T` implements `Send` if it is safe to move `T` across threads.
- `T` implements `Sync` if it is safe to share `T` between threads.

trait `Send` and `Sync` are used to ensure that data is safe to be shared across threads. 5-59

5-59

```

1. use std::thread;
2. fn main() {
3.     let mut data=vec![1, 2, 3];
4.     for i in 0..3 {
5.         thread::spawn(move || {
6.             data[i] +=1;
7.         });
8.     }
9.     thread::sleep_ms(50);
10. }

```

5-59

```
error[E0382]: capture of moved value: `data`
7 |         thread::spawn(move|| {
  |                        ----- value moved (into closure) here
8 |             data[i] +=1;
  |             ^^^^ value captured here after move
```

```
data5-59
data
```

```
RustArcTMutexTRwLockT
Atomic
```

```
· ArcTRcT
· MutexT
· RwLockTRefCellTreaderwriter
```

```
· Atomic AtomicBoolAtomicIsizeAtomicUsize
AtomicPtr4 AtomicPtr
CellT
```

```
11
```

5.8

```
Rust5-60
```

```
5-60
```

```

1. fn foo<'a>(x: &'a str, y: &'a str) -> &'a str {
2.     if x.len() % 2 == 0 {
3.         x
4.     } else {
5.         y
6.     }
7. }
8. fn main(){
9.     let x = String::from("hello");
10.    let z;
11.    let y = String::from("world");
12.    z = foo(&x, &y);
13.    println!("{:?}", z);
14. }

```

5-60 12 &y 10 11

Rust 5-60 x y z x z y 12 &y foo z y z y x y z

Rust Rust 2018 **Non-Lexical Lifetime** **NLL**

MIR

1 Rust HIR HIR MIR MIR LLVM IR AST HIR HIR AST AST MIR

MIR **Control Flow Graph** CFG
DAG MIR
NLL

MIR

• **Basic Block** **bb**

➤ Statement

➤ Terminator

• **Local**

_1 _0

• **Place** _1_1.f

• **RValue**

MIR 5-61

5-61 MIR

```
1. fn main() {  
2.     let a = 1;  
3.     a + 2;  
4. }
```

5-61 play.rust-lang.org MIR 5-62

5-62 MIR

```

1.  // MIR 解释
2.  fn main() -> () {
3.      let mut _0: (); // 返回值
4.      scope 1 {        // 第一个变量产生的顶级作用域，会包裹其他变量
5.      }
6.      scope 2 {        // a 自己的作用域
7.          let _1: i32;
8.      }
9.      let mut _2: i32; // 临时值
10.     let mut _3: i32;
11.     let mut _4: (i32, bool);
12.     bb0: {            // 基础块
13.         StorageLive(_1); // 语句，代表活跃，LLVM用它来分配栈空间
14.         _1 = const 1i32; // 赋值
15.         StorageLive(_3);
16.         _3 = _1;        // 使用临时变量
17.         // 执行 Add 操作，具有防溢出检查，
18.         // 其中 move 代表 move 语义，编译器会自己判断是不是 Copy
19.         _4 = CheckedAdd(move _3, const 2i32);
20.         // 断言，溢出时抛出错误，并且流向 bb1 块。此为终止句
21.         assert(!move (_4.1: bool), "attempt to add with overflow")
22.         -> bb1;
23.     }
24.     bb1: {            // 基础块
25.         _2 = move (_4.0: i32); // 赋值，右值默认是 move
26.         StorageDead(_3);    // 语句，代表不活跃，LLVM用它来分配栈空间
27.         StorageDead(_1);
28.         return;            // 返回
29.     }
30. }

```

5-625-61MIR
 bb0bb1StorageLiveStorageDead
 LLVMLLVMStack

5-64

5-64

```
1. struct List<T> {
2.     value: T,
3.     next: Option<Box<List<T>>>,
4. }
5. fn to_refs<T>(mut list: &mut List<T>) -> Vec<&mut T> {
6.     let mut result = vec![];
7.     loop {
8.         result.push(&mut list.value);
9.         if let Some(n) = list.next.as_mut() {
10.             list = n;
11.         } else {
12.             return result;
13.         }
14.     }
15. }
16. fn main() {}
```

5-64

Rust 2018

NLL

5-65

5-65

```
1. fn main() {
2.     let mut x = vec![1];
3.     x.push(x.pop().unwrap());
4. }
```

5-65

5-65


```
error[E0506]: cannot write to `x` while borrowed
|
|   x.push(x.pop().unwrap());
|   - ---- ^^^^^^^^^^^^^^^^^^^^^^^
|
|   | |   write to `x` occurs here, while borrow is still in active use
|   | borrow is later used here, during the call
|           |           `x` borrowed here
```

NLL

5.9 □□

```

    RustRust
    Rust
    RustCopy trait

```

[illegible]

Non Lexical Lifetime

Rust

Rust

```

    Rust Rc T
    Rust Rc T Weak T Rust
    Cell T RefCell T Rust
    Rust Cow T

```

このドキュメントは Rust の NLL に関するものです。
NLL は Rust 2018 の新機能です。

Rust 2018 の NLL は、Rust の NLL に関するものです。
NLL は Rust の NLL に関するものです。

Rust の NLL に関する Rust の

[1] <https://github.com/rust-lang/rfcs/blob/master/text/2094-nll.md>

[2] <https://rust-lang-nursery.github.io/rustc-guide/mir/index.html>

第6章 函数式编程

本章主要介绍函数式编程

Rust 语言支持函数式编程，包括 `impl` 和 `trait` 等特性。Rust 语言还支持 `Rust` 语言，包括 `Rust` 语言。

本章主要介绍 LISP、Python、Ruby、Haskell、CPU、Elixir、Scala、Swift、LISP、Haskell、algebraic data type、C++、Java、Rust 等语言。

本章主要介绍 Rust 语言。

6.1 函数

本章主要介绍函数，包括 `fn` 函数。

本章主要介绍 **6-1** 函数。

```

1. // 函数定义形式
2. fn func_name(arg1: u32, arg2: String) -> Vec<u32> {
3.     /* 函数体*/
4. }
5. // 利用 Raw identifier 将语言关键字用作函数名 (Rust 2018 版本)
6. fn r#match(needle: &str, haystack: &str) -> bool {
7.     haystack.contains(needle)
8. }
9.
10. fn main() {
11.     assert!(r#match("foo", "foobar"));
12. }

```

6-1 **fn** 标识符与 **snake_case** 标识符
 Rust 标识符与 C 标识符 Rust 标识符与 C 标识符 6-1 6

Rust 2018
 Raw Identifier **r** 标识符与 C 标识符
 FFI C Rust 标识符与 C 标识符 6-1 6

Copy 标识符与 C 标识符
 Rust 标识符与 C 标识符 6-1 6

Rust 标识符与 C 标识符 **mut**
 6-2 标识符与 C 标识符 **mut**

6-2 标识符与 C 标识符 **mut**

```

1. fn modify(mut v: Vec<u32>) -> Vec<u32> {
2.     v.push(42);
3.     v
4. }
5. fn main() {
6.     let v = vec![1,2,3];
7.     let v = modify(v);
8.     println!("{:?}", v);
9. }

```

6-2 modify 6-2
 main 6-2 v Vec<u32> modify
 1 modify mut
 main v mut

6-3 mut

6-3 mut

```

1. fn modify(v: &mut [u32]) {
2.     v.reverse();
3. }
4. fn main() {
5.     let mut v = vec![1,2,3];
6.     modify(&mut v);
7.     println!("{:?}", v); // [3, 2, 1]
8. }

```

6-3 modify &mut [u32]
 mut main 5 v
 mut

6.1.1

variable shadow 6-3 modify
 error[E0428]: the name `modify` is defined multiple times

Figure 6-5: A function that takes a reference to a mutable variable and returns a reference to it. The function `ref_mut` takes a mutable reference to a mutable variable and returns a mutable reference to it. The `main` function calls `ref_mut` with a mutable reference to a mutable variable `s`.

Figure 6-6: A function that takes a mutable reference to a mutable variable and returns a mutable reference to it.

Figure 6-6: A function that takes a mutable reference to a mutable variable and returns a mutable reference to it.

```
1. fn foo(_: i32) {  
2.     // ...  
3. }  
4. fn main() {  
5.     foo(3);  
6. }
```

Figure 6-7: A function that takes a mutable reference to a mutable variable and returns a mutable reference to it. The function `swap` takes a mutable reference to a mutable variable and returns a mutable reference to it. The `main` function calls `swap` with a mutable reference to a mutable variable `t`.

Rust `let` statement can be used to declare a mutable variable. The `let` statement can be used to declare a mutable variable. The `let` statement can be used to declare a mutable variable.

Figure 6-7: A function that takes a mutable reference to a mutable variable and returns a mutable reference to it.

```
1. fn swap((x, y): (&str, i32)) -> (i32, &str){  
2.     (y, x)  
3. }  
4. fn main() {  
5.     let t = ("Alex", 18);  
6.     let t = swap(t);  
7.     assert_eq!(t, (18, "Alex"));  
8. }
```

Figure 6-7: A function that takes a mutable reference to a mutable variable and returns a mutable reference to it. The function `swap` takes a mutable reference to a mutable variable and returns a mutable reference to it. The `main` function calls `swap` with a mutable reference to a mutable variable `t`.

6.1.3 `let` statement

Rust `let` statement can be used to declare a mutable variable. The `let` statement can be used to declare a mutable variable. The `let` statement can be used to declare a mutable variable.

Figure 6-8: A function that takes a mutable reference to a mutable variable and returns a mutable reference to it.

```

1. fn addsub(x: isize, y: isize) -> (isize, isize) {
2.     (x + y, x - y)
3. }
4. fn main() {
5.     let (a, b) = addsub(5, 8);
6.     println!("a: {:?}, b: {:?}", a, b);
7. }

```

6-8 `addsub` `main` `let` `a` `b`

Rust `return` `addsub` `return` 6-9

6-9 `return`

```

1. fn gcd(a: u32, b: u32) -> u32 {
2.     if b == 0 { return a; }
3.     return gcd(b, a % b);
4. }
5. fn main() {
6.     let g = gcd(60, 40);
7.     assert_eq!(20, g);
8. }

```

6-9 `gcd` `a%b` `0` `b` `a` `b` `0` `a` `gcd` `if-else`

2 `diverging function`

6.1.4

Rust 6-10

6-10


```

1. use std::ops::Mul;
2. fn square<T: Mul<T, Output=T>>(x: T, y: T) -> T {
3.     x * y
4. }
5. fn main() {
6.     let a: i32 = square(37, 41);
7.     let b: f64 = square(37.2, 41.1);
8.     assert_eq!(a, 1517);
9.     assert_eq!(b, 1528.92); // 浮点数执行结果可能有所差别
10. }

```

6-10 为 `square` 函数实现，`T` 为 `Mul` trait 实现 `Mul` 的任意类型。
 `Mul` trait 定义在 `std::ops` 模块中，`Output=T` 表示
 `main` 函数中 `i32` 和 `f64` 类型调用 `square` 函数

6-11 为 `turbofish` 函数实现，6-11 为

6-11 turbofish

```

1. use std::ops::Mul;
2. fn square<T: Mul<T, Output = T>>(x: T, y: T) -> T {
3.     x * y
4. }
5. fn main() {
6.     let a = square::<u32>(37, 41);
7.     let b = square::<f32>(37.2, 41.1);
8.     assert_eq!(a, 1517);
9.     assert_eq!(b, 1528.9199);
10. }

```

6-11 为 `turbofish` 函数实现，6-11 为
 `a` `b`

6.1.5

Rust 的 `struct` 和 `impl` 是定义结构体和实现的方法。结构体是 Rust 中最基本的类型之一，它用于组织数据。实现则是为结构体定义的方法，用于操作数据。

6-12 定义 `User` 结构体

6-12 定义 `User` 结构体

```
1.  #[derive(Debug)]
2.  struct User {
3.      name: &'static str,
4.      avatar_url: &'static str,
5.  }
6.  impl User {
7.      fn show(&self) {
8.          println!("name: {:?} ", self.name);
9.          println!("avatar: {:?} ", self.avatar_url);
10.     }
11. }
12. fn main() {
13.     let user = User {
14.         name: "Alex",
15.         avatar_url: "https://avatar.com/alex"
16.     };
17.     // User::show(&user)
18.     user.show();
19. }
```

6-12 定义 `User` 结构体 `name` `avatar_url` `impl` `User` `show` `&self` `self` `User` `&self`

`main` `show` 18 `user` `show` `user` `show` `user.show` `User` `show` `&user` 7

6.1.6 小结


```

1. fn hello(){
2.     println!("hello function pointer");
3. }
4. fn main(){
5.     let fn_ptr: fn() = hello;
6.     println!("{:p}", fn_ptr); // 0x562bacfb9f80
7.     let other_fn = hello;
8.     // println!("{:p}", other_fn); // 非函数指针
9.     hello();
10.    other_fn();
11.    fn_ptr();
12.    (fn_ptr)();
13. }

```

6-14 5 1 let 6 fn hello 6 fn_ptr

7 let 8 other_fn

error[E0277]: the trait bound `fn() {hello}: std::fmt::Pointer` is not satisfied

```

8 |     println!("{:p}", other_fn);
  |                               ^^^^^^^^^ the trait `std::fmt::Pointer` is not
  |                               implemented for `fn() {hello}`

```

other_fn fn {hello} hello other_fn 10

6-13 math op fn i32 i32 - i32 main math sum product

type 6-15

6-15 type

```

1. type MathOp = fn(i32, i32) -> i32;
2. fn math(op: MathOp, a: i32, b: i32) -> i32{
3.     println!("{:p}", op);
4.     op(a, b)
5. }

```

6-16

6-16

```

1. type MathOp = fn(i32, i32) -> i32;
2. fn math(op: &str) -> MathOp {
3.     fn sum(a: i32, b: i32) -> i32 {
4.         a + b
5.     }
6.     fn product(a: i32, b: i32) -> i32 {
7.         a * b
8.     }
9.     match op {
10.        "sum" => sum,
11.        "product" => product,
12.        _ => {
13.            println!(
14.                "Warning: Not Implemented {:?} operator, Replace with
sum",
15.                op
16.            );
17.            sum
18.        }
19.    }
20. }

```

```

21. fn main() {
22.     let (a, b) = (2, 3);
23.     let sum = math("sum");
24.     let product = math("product");
25.     let div = math("div");
26.     assert_eq!(sum(a, b), 5);
27.     assert_eq!(product(a, b), 6);
28.     assert_eq!(div(a, b), 5);
29. }

```

6-16 `math` `match`
`sum` `sum` `product` `product`
`match` `sum` `product`
 25 `div` `warning` `sum` `div`

`math`
 6-17

6-17

```

1. fn sum(a: i32, b: i32) -> i32 {
2.     a + b
3. }
4. fn product(a: i32, b: i32) -> i32 {
5.     a * b
6. }
7. type MathOp = fn(i32, i32) -> i32;
8. fn math(op: &str, a: i32, b: i32) -> MathOp {
9.     match op {
10.         "sum" => sum(a, b),
11.         _ => product(a, b)
12.     }
13. }
14. fn main() {
15.     let (a, b) = (2, 3);
16.     let sum = math("sum", a, b);
17. }

```

6-17 `math` `match` `sum` `a` `b` `product` `a` `b` `i32` `MathOp`

6-18

6-18 1

```

1. fn counter() -> fn(i32) -> i32 {
2.     fn inc(n: i32) -> i32 {
3.         n + 1
4.     }
5.     inc
6. }
7. fn main() {
8.     let f = counter();
9.     assert_eq!(2, f(1));
10. }

```



```

1. fn counter(i: i32) -> Box<Fn(i32) -> i32> {
2.     Box::new(move |n: i32| n + i )
3. }
4. fn main() {
5.     let f = counter(3);
6.     assert_eq!(4, f(1));
7. }

```

6-20 counter Box Trait impl Trait impl Fn i32 -> i32 Box Trait

2 |n i32| n+i i n 5
i
move i
i

Fn i32 -> i32 F Fn
fn i32 -> i32 trait

main 5 f counter 3
counter 3 6 f 1 4

- -
- Rust

6.2.1

Rust Ruby lambda 6-21
6-21

```

1. fn main() {
2.     let add = |a: i32, b: i32| -> i32 { a + b };
3.     assert_eq!(add(1, 2), 3);
4. }

```

编译选项 编译选项编译选项编译选项编译选项编译选项编译选项编译选项编译选项
编译选项编译选项编译选项编译选项编译选项编译选项编译选项编译选项

```
let add = |a, b| -> i32 { a + b };
```

编译选项编译选项编译选项编译选项编译选项编译选项编译选项编译选项

```
let add = |a, b| a + b ;
```

编译选项编译选项编译选项编译选项编译选项编译选项编译选项编译选项

```
let (a, b) = (1, 2);
```

```
let add = || a + b ;
```

编译选项编译选项编译选项编译选项6-22编译

编译6-22编译选项编译选项编译

```
1. fn val() -> i32 { 5 }
2. fn main(){
3.     let add = |a: fn() -> i32, (b, c)| (a)() + b + c;
4.     let r = add(val, (2, 3));
5.     assert_eq!(r, 10);
6. }
```

编译6-22编译3编译选项编译选项编译选项编译选项编译选项编译选项编译选项编译选项
编译选项编译选项编译选项编译选项Rust编译选项编译选项编译选项编译选项i32编译选项
编译选项编译选项

编译选项编译选项编译选项编译选项编译选项编译选项编译选项编译选项6-23编译

编译6-23编译选项编译选项编译选项编译选项编译

```
1. fn main(){
2.     let c1 = || {};
3.     let c2 = || {};
4.     let v = [c1, c2];
5. }
```

编译6-23编译选项编译选项编译选项编译选项编译选项编译选项编译选项编译选项
编译选项编译选项编译选项编译选项

error[E0308]: mismatched types

```
5 |     let v = [c1, c2];
```

```
|               ^^ expected closure, found a different closure
```

6.2.2 实验结果

11

6-24

```
1. fn main(){
2.     let c1 : () = || {println!("i'm a closure")};
3. }
```

□□□□6-24□□□□□□□□

[illegible]

```
found type `[closure@src/main.rs:3:19: 3:49]`
```

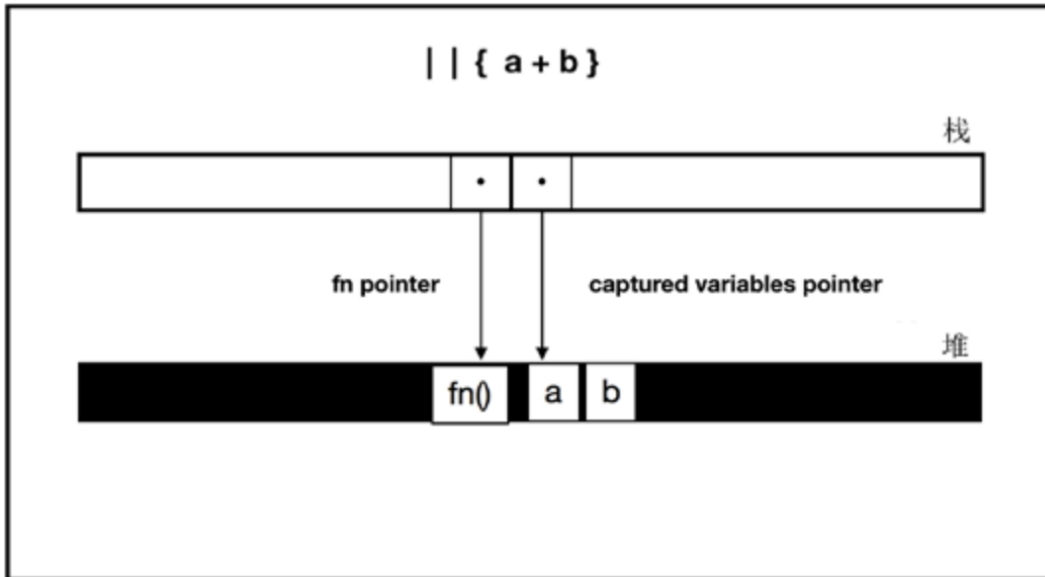
```

[closure@src/main.rs:3:19:3:49]  Rust

```

Rust

6-1 Rust
 Rust
Boxed
 Rust LLVM LLVM



6-1 闭包与函数指针

Rust 中，闭包（Closure）和函数指针（Function Pointer）是两个不同的概念。闭包是指一个函数及其捕获的变量的集合。而函数指针则是指向一个函数的指针。Unboxed 是指 Rust 中的未装箱类型。

闭包的内存布局如下：

- 函数指针（fn pointer）
- 捕获的变量（captured variables）
- 闭包的内存布局如下：
self & self & mut self

闭包的内存布局如下：
trait 闭包的内存布局如下：
a b c d

```
Fn::call(&a, (b, c, d))
```

```
FnMut::call_mut(&mut a, (b, c, d))
```

```
FnOnce::call_once(a, (b, c, d))
```

Rust 中的 trait 闭包 Fn FnMut FnOnce Rust 中的 trait 闭包 6-25

6-25 Fn FnMu FnOnce


```

26. }
27. fn call_it_mut<F: FnMut() -> u32>(f: &mut F) -> u32 {
28.     f()
29. }
30. fn call_it_once<F: FnOnce() -> u32>(f: F) -> u32 {
31.     f()
32. }
33. fn main() {
34.     let env_var = 1;
35.     let mut c = Closure { env_var: env_var };
36.     c();
37.     c.call(());
38.     c.call_mut(());
39.     c.call_once(());
40.     let mut c = Closure { env_var: env_var };
41.     {
42.         assert_eq!(3, call_it(&c));
43.     }
44.     {
45.         assert_eq!(3, call_it_mut(&mut c));
46.     }
47.     {
48.         assert_eq!(3, call_it_once(c));
49.     }
50. }

```

6-26 1 feature **[feature**
unboxed_closures**fn_traits]** Nightly

2 Closure
 FnOnce FnMut Fn trait

24 32 call_it call_it_mut call_it_once
 FnOnce FnMut Fn trait Closure

main 35 Closure env_var
trait 36 c

6-27

6-27

1. call it Fn()
2. call it Fn()
3. call it FnMut()
4. call it FnOnce()
5. call it Fn()
6. call it FnMut()
7. call it FnOnce()

6-27 1 6-26 36 c Fn
trait call " "

extern "rust-call" fn call(&self, args: ()) -> u32

extern fn ABI Application Binary
Interface Rust rust-call ABI
Fn FnMut FnOnce trait
rust-call ABI

rust-call ABI 6-26 1
unboxed_closures

6-26 37 39 call call_mut call_once
args
6-27 2 4

6-26 40 Closure 39 call_once
c call_once self

6-26 41 49 call_it call_it_mut call_it_once
trait trait

F: Fn() -> u32
F: FnMut() -> u32
F: FnOnce() -> u32

□□□□□□□□6-27□□5□□□7□□□□□□□□
 □□□□6-26□□□□□□□□□□□□□□□□6-28□□□
 □□□□**6-28**□□□□□□**6-26**□□□□□□□□

```
1. fn main() {
2.     let env_var = 1;
3.     let c = || env_var + 2;
4.     assert_eq!(3, c());
5. }
```

6-28 6-26 trait
 Closurec

6-26 Rust

trait trait
6-29

6-29

```
1. fn main(){
2.     let env_var = 1;
3.     let c : Box<Fn() -> i32>= Box::new(||{ env_var + 2});
4.     assert_eq!(3, c());
5. }
```

```

Box\Fn-i32 trait
trait

```

6.2.3 实验结果

```

trait FnOnce {
    fn call(self) -> ();
}
trait FnMut {
    fn call(self) -> ();
}
trait Fn {
    fn call(&self) -> ();
}

```

```
trait trait trait trait
```

[illegible]

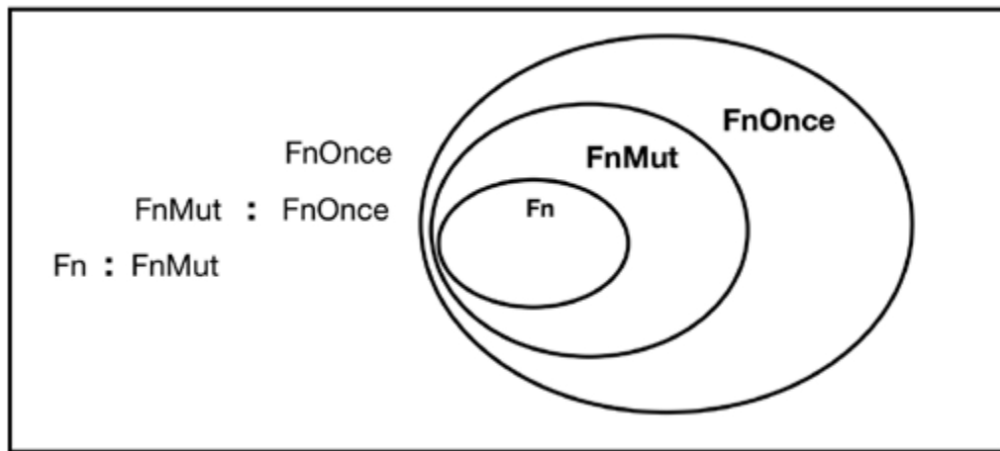
```
· FnMut [usize; 64]
[usize; 64]&mut self]
```

· **FnOnce** 只能被调用一次，调用后返回 `self`

5. 只能被调用一次，调用后返回 `self`

- 只能被调用一次，调用后返回 `&T`
- 只能被调用一次，调用后返回 `&mut T`
- 只能被调用一次，调用后返回 `&mut T`

6-2. `Fn`、`FnMut`、`FnOnce` 的关系



6-2. `Fn`、`FnMut`、`FnOnce` 的关系

6-2. `Fn`、`FnMut`、`FnOnce` 的关系
`FnOnce` 是 `Fn` 和 `FnMut` 的上界，`FnMut` 是 `Fn` 的上界。
`FnOnce` 只能被调用一次，调用后返回 `self`。
`FnMut` 可以被调用多次，调用后返回 `&mut T`。
`Fn` 可以被调用多次，调用后返回 `&T`。

只能被调用一次 **Fn**

只能被调用一次 6-30

6-30 只能被调用一次 **Fn**

```
1. fn main() {  
2.     let s = "hello";  
3.     let c = ||{ println!("{:?}", s) };  
4.     c();  
5.     c();  
6.     println!("{:?}", s);  
7. }
```

6-30 s c
s 4 5 c 6 println s
c 6
println s 3 s

c Fn trait
Fn FnMut FnOnce 6-30
trait Fn FnMut FnOnce 6-31
6-31 6-30 trait

```

1.  #![feature(unboxed_closures, fn_traits)]
2.  struct Closure<'a> {
3.      env_var: &'a u32
4.  }
5.  impl<'a> FnOnce<()> for Closure<'a> {
6.      type Output = ();
7.      extern "rust-call" fn call_once(self, args: ()) -> () {
8.          println!("{:?}", self.env_var);
9.      }
10. }
11. impl<'a> FnMut<()> for Closure<'a> {
12.     extern "rust-call" fn call_mut(&mut self, args: ()) -> () {
13.         println!("{:?}", self.env_var);
14.     }
15. }
16. impl<'a> Fn<()> for Closure<'a> {
17.     extern "rust-call" fn call(&self, args: ()) -> () {
18.         println!("{:?}", self.env_var);
19.     }
20. }
21. fn main(){
22.     let env_var = 42;
23.     let mut c = Closure{env_var: &env_var};
24.     c(); //42
25.     c.call_mut(); // 42
26.     c.call_once(); // 42
27. }

```

6-31 Closure a main 24
 c Fn call Fn FnMut
 FnOnce 25 26 call_mut call_once
 6-30 call_mut call_once
 6-32

6-32 Fn call_mut call_once

```

1.  #![feature(fn_traits)]
2.  fn main() {
3.      let s = "hello";
4.      let mut c = ||{ println!("{:?}", s) };
5.      c(); // "hello"
6.      c(); // "hello"
7.      c.call_mut(()); // "hello"
8.      c.call_once(()); // "hello"
9.      c; // "hello"
10.     println!("{:?}", s); // "hello"
11. }

```

6-32 1 **feature fn_traits** trait call call_mut call_once
 6-30

4 mut call_mut

5 Fn call 6 c
 7 call_mut
 8 call_once c s
 Copy FnOnce Copy call_once
 c 9 c c
 call_once

10 s c call_mut call_once

FnOnce

6-33

6-33 FnOnce

```

1. fn main() {
2.     let s = "hello".to_string();
3.     let c = || s;
4.     c();
5.     // c(); // error: use of moved value: `c`
6.     // println!("{:?}", s); // error: use of moved value: `s`
7. }

```

6-33 `s` `String` 5 `c`
`c` 6 4 `c`
`s` `c`
`trait` `self`
`FnOnce` 6 `s` `c` `s`

`FnOnce`
`FnOnce` `call` `call_mut`
 6-34

6-34 `FnOnce` `call` `call_mut`

```

1. #![feature(fn_traits)]
2. fn main() {
3.     let mut s = "hello".to_string();
4.     let c = || s;
5.     c();
6.     // error: expected a closure that implements the `FnMut` trait,
7.     //         but this closure only implements `FnOnce`
8.     // c.call(());
9.     // error: expected a closure that implements the `FnMut` trait,
10.    //         but this closure only implements `FnOnce`
11.    // c.call_mut(());
12.    // c(); // error: use of moved value: `c`
13.    // println!("{:?}", s); // error use of moved value: `s`
14. }

```

6-34 3 `mut`
`call_mut`

move 6-37

6-37 move

```
1. fn call<F: FnOnce()>(f: F) { f() }
2. fn main() {
3.     // 未使用 move
4.     let mut x = 0;
5.     let incr_x = || x += 1;
6.     call(incr_x);
7.     // call(incr_x); // ERROR: `incr_x` moved in the call above.
8.     // 使用 move
9.     let mut x = 0;
10.    let incr_x = move || x += 1;
11.    call(incr_x);
12.    call(incr_x);
13.    // 对移动语义类型使用 move
14.    let mut x = vec![];
15.    let expend_x = move || x.push(42);
16.    call(expend_x);
17.    // call(expend_x); // ERROR: use of moved value: `expend_x`
18. }
```

6-37 call FnOnce

4 7 incr_x move x call incr_x

9 12 incr_x move call

14 17 expend_x move x call expend_x

6-37 move Copy/Clone Copy/Clone

FnMut

FnMut6-38

6-38 FnMut

```
1. fn main() {
2.     let mut s = "rush".to_string();
3.     {
4.         let mut c = ||{ s += " rust" };
5.         c();
6.         c();
7.         // error: cannot borrow `s` as immutable
8.         //         because it is also borrowed as mutable
9.         // println!("{:?}", s);
10.    }
11.    println!("{:?}", s);
12. }
```

```

    6-36smutss4ccmutFnMutfn_mut&mut self

```

[illegible]

FnMut FnOnce Fn 6-39

6-39 FnMut

```

1. #![feature(fn_traits)]
2. fn main () {
3.     let mut s = "rush".to_string();
4.     {
5.         let mut c = || s += " rust";
6.         c();
7.         // error: expected a closure that implements the `Fn` trait,
8.         //         but this closure only implements `FnMut`
9.         // c.call(());
10.        c.call_once(());
11.        // error: cannot borrow `s` as immutable
12.        //         because it is also borrowed as mutable
13.        // println!("{:?}",s);
14.    }
15.    println!("{:?}",s); // "rush rust rust"
16. }

```

6-39 9 call
 FnMut 10 call_once

Fn

Fn 6-40

6-40 Fn

```

1. fn main() {
2.     let c = ||{ println!("hhh") };
3.     c();
4.     c();
5. }

```

6-40 c mut
 &self
 Fn

Fn

Fn

• Fn

- 閉包を返す関数
- move 関数 Fn
- FnMut
- 閉包を返す関数
- move 関数 FnOnce
- move 関数 Fn
- FnMut
- move 関数 Copy/Clone
Copy/Clone
trait

6.2.4 閉包

JavaScript、Python、Ruby は閉包をサポートしている。GC は CPU を消費する。Rust は trait をサポートしている。Rust は C++ の閉包をサポートしている。

Rust は trait をサポートしている。Rust は C++ の閉包をサポートしている。6-41

6-41 trait

```
1. fn boxed_closure(c: &mut Vec<Box<Fn()>>){
```

```

2.     let s = "second";
3.     c.push(Box::new(|| println!("first")));
4.     c.push(Box::new(move || println!("{}", s)));
5.     c.push(Box::new(|| println!("third")));
6. }
7. fn main(){
8.     let mut c: Vec<Box<Fn()>> = vec![];
9.     boxed_closure(&mut c);
10.    for f in c {
11.        f(); // first / second / third
12.    }
13. }

```

6-41 8 Vec Box Fn Box Fn trait Box T trait 3 trait vtable 4 s iter_call move s s s

boxed_closure escape closure boxed_closure non-escape closure

Rust Ruby Python Rust Ruby any

Rust

```

v.any(|&x| x == 3);

```

Ruby

```

v.any?{|i| i == 3}

```

Rust Ruby

Figure 6-43: A closure example

```
1. fn call<F>(closure: F) -> i32
2. where F: Fn(i32) -> i32
3. {
4.     closure(1)
5. }
6. fn counter(i: i32) -> i32 { i+1 }
7. fn main(){
8.     let result = call(counter);
9.     assert_eq!(2, result);
10. }
```

Figure 6-43: A closure example. The `call` function takes a closure `closure` of type `Fn(i32) -> i32` and calls it with the argument `1`. The `counter` function is a closure that takes an `i32` and returns `i+1`.

Figure 6-42: A trait example. The `where` clause is used to restrict the type `Self` to be `Sized`. The `Any` trait is used to check if the type `Self` is `Any`.

Figure 6-42: A trait example. The `impl` block implements the `Vec` trait for `u32`. The `any` method is used to check if the type `u32` is `Any`. The `true` and `false` values are used to indicate the result of the check.

The `main` function is used to test the `Vec` trait implementation. The `any` method is used to check if the type `u32` is `Any`. The `v` variable is used to store the result of the `any` method. The `Rust` variable is used to store the result of the `any` method.

The `trait` block is used to define the `Vec` trait. The `6-44` figure shows the `trait` block. The `6-44` figure shows the `trait` block.

```

1. trait Any {
2.     fn any(&self, f: &(Fn(u32) -> bool)) -> bool;
3. }
4. impl Any for Vec<u32> {
5.     fn any(&self, f: &(Fn(u32) -> bool)) -> bool {
6.         for &x in self.iter() {
7.             if f(x) {
8.                 return true;
9.             }
10.        }
11.        false
12.    }
13. }
14. fn main(){
15.     let v = vec![1,2,3];
16.     let b = v.any(&|x| x == 3);
17.     println!("{:?}", b);
18. }

```

6-44 trait
C++ callback

Rust Web Rocket
6-45

6-45 Rocket

13 AdHoc on_request F trait Fn &mut Request &Data + Send + Sync + static static 6-46

6-46 static

```
1. fn main(){
2.     let s = "hello";
3.     let c: Box<Fn() + 'static> = Box::new( move||{ s;});
4. }
```

6-46 3 move Fn static move

6-45 21 AdHoc Fairing trait on_request if let

trait trait 6-47

6-47

```
1. fn square() -> Box<Fn(i32) -> i32> {
2.     Box::new(|i| i*i )
3. }
4. fn main(){
5.     let square = square();
6.     assert_eq!(4, square(2));
7.     assert_eq!(9, square(3));
8. }
```

6-47 trait Box Fn i32 - i32 main

6-47 Fn FnOnce 6-48

6-48 FnOnce

6-48

the size of `std::ops::FnOnce(i32) -> i32` cannot be statically determined

```

    FnOnce [Box] FnOnce [Box]
    Box [ClosureStruct] [ClosureStruct]
    FnOnce [call_once] [self] [Box]
    self [Box] self [self]
    [self] Fn [FnMut]
    &Box [self] &mut Box [self] Rust
    6-49

```

```
1.  #![feature(fnbox)]
2.  use std::boxed::FnBox;
3.  fn square() -> Box<FnBox(i32) -> i32> {
4.      Box::new( |i| {i*i } )
5.  }
6.  fn main(){
7.      let square = square();
8.      assert_eq!(4, square(2));
9.  }
```

6-49 1 **feature fnbox** use boxed FnBox trait FnOnce FnBox FnBox “” 6-50 FnBox

6-50 FnBox

```
1. #[rustc_paren_sugar]
2. pub trait FnBox<A> {
3.     type Output;
4.     fn call_box(self: Box<Self>, args: A) -> Self::Output;
5. }
6. impl<A, F> FnBox<A> for F
7.     where F: FnOnce<A>
8.     {
9.         type Output = F::Output;
10.        fn call_box(self: Box<F>, args: A) -> F::Output {
11.            self.call_once(args)
12.        }
13.    }
14. impl<'a, A, R> FnOnce<A> for Box<FnBox<A, Output = R> + 'a> {
15.     type Output = R;
16.     extern "rust-call" fn call_once(self, args: A) -> R {
17.         self.call_box(args)
18.     }
19. }
```

6-50 FnBox FnBox **rustc_paren_sugar** trait call_box Fn FnMut FnOnce self Box Self trait self &self &mut self

- Self self Self
- &self self &Self
- &mut self self &mut Self

```
    self.SomeType::Self::SomeType  
    Box::T::self::Box::Self::Self  
    Box::T::DerefMove::5
```

```
Self::Box::self::call_box::call_once::Box  
FnBox::FnOnce::“”::FnBox  
FnOnce::Rust::FnBox
```

```
Rust::trait  
Rust::impl Trait  
trait6-51
```

6-51 impl Trait

```
1. fn square() -> impl FnOnce(i32) -> i32 {  
2.     |i| {i*i}  
3. }  
4. fn main(){  
5.     let square = square();  
6.     assert_eq!(4, square(2));  
7. }
```

6-51 impl Trait Rust 2018 impl
Trait trait

```
2impl FnOnceu8u8-u8impl  
traitFnOnce trait
```

6.2.5

6-52

6-52 trait trait

```

1. use std::fmt::Debug;
2. trait DoSomething<T> {
3.     fn do_sth(&self, value: T);
4. }
5. impl<'a, T: Debug> DoSomething<T> for &'a usize {
6.     fn do_sth(&self, value: T) {
7.         println!("{:?}", value);
8.     }
9. }
10. fn foo<'a>(b: Box<DoSomething<&'a usize>>) {
11.     let s: usize = 10;
12.     b.do_sth(&s)
13. }
14. fn main(){
15.     let x = Box::new(&2usize);
16.     foo(x);
17. }

```

6-52 DoSomething T trait do_sth &usize trait

10 13 foo b trait Box DoSomething &usize b do_sth s do_sth foo

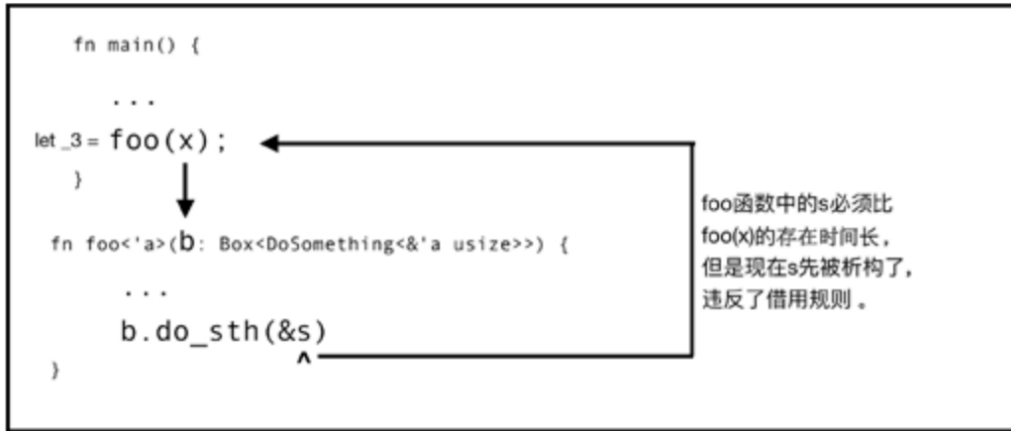
main Box &usize x foo x

```

error[E0597]: `s` does not live long enough
12 |     b.do_sth(&s)
    |               ^ does not live long enough
13 | }
    | - borrowed value only lives until here

```

s foo &s Rust 6-3



6-3 6-52

6-52 10 foo trait Box DoSomething &usize &usize 6-52 15 main foo foo 6-52 b.do_sth &s s main foo x main 16 let_3=foo x 5

foo trait Box DoSomething &usize &usize 6-52 15 main foo foo 6-52

Rust Higher-Ranked Lifetime trait Higher-Ranked Trait Bound HRTB for 6-53

6-53 for

```

1. use std::fmt::Debug;
2. trait DoSomething<T> {
3.     fn do_sth(&self, value: T);
4. }
5. impl<'a, T: Debug> DoSomething<T> for &'a usize {
6.     fn do_sth(&self, value: T) {
7.         println!("{:?}", value);
8.     }
9. }
10. fn bar(b: Box<for<'f> DoSomething<&'f usize>>) {
11.     let s: usize = 10;
12.     b.do_sth(&s);
13. }
14. fn main(){
15.     let x = Box::new(&2usize);
16.     bar(x);
17. }

```

6-53 10 bar for
 f DoSomething & f usize
 for
 for DoSomething & f usize f for f
 for f bar DoSomething & f usize
 for

trait trait
 6-54

6-54

```

1. struct Pick<F> {
2.     data: (u32, u32),
3.     func: F,
4. }
5. impl<F> Pick<F>
6. where F: Fn(&(u32, u32)) -> &u32
7. {
8.     fn call(&self) -> &u32 {
9.         (self.func) (&self.data)
10.    }
11. }
12. fn max(data: &(u32, u32)) -> &u32 {
13.    if data.0 > data.1{
14.        &data.0
15.    }else{
16.        &data.1
17.    }
18. }
19. fn main() {
20.    let elm = Pick { data: (3, 1), func: max };
21.    println!("{}", elm.call());
22. }

```

6-54 Pick struct data func

5 11 Pick call F Fn & u32 u32 - &u32 trait

trait

9 Pick call &self
 6-52 6-53 call
 self.func call
 call


```
fn call<'a>(&'a self) -> &'a u32 {
    (self.func) (&self.data)
}
```

6-55

6-55

```
1. struct Pick<F> {
2.     data: (u32, u32),
3.     func: F,
4. }
5. impl<F> Pick<F>
6. where F: for<'f> Fn(&'f (u32, u32)) -> &'f u32, // 显式指定
7. {
8.     fn call(&self) -> &u32 {
9.         (self.func) (&self.data)
10.    }
11. }
12. fn max(data: &(u32, u32)) -> &u32 {
13.     if data.0 > data.1{
14.         &data.0
15.     }else{
16.         &data.1
17.     }
18. }
19. fn main() {
20.     let elm = Pick { data: (3, 1), func: max };
21.     println!("{}", elm.call());
22. }
```

6-55

for

6.3

Rust 6-42 any for

Iterator Cursor

6.3.1

External Iterator Internal Iterator

Active Iterator next Python Java C++

Ruby each

1.0 Rust

6-56

6-56

```
1. trait InIterator<T: Copy> {
2.     fn each<F: Fn(T) -> T>(&mut self, f: F);
3. }
4. impl<T: Copy> InIterator<T> for Vec<T> {
5.     fn each<F: Fn(T) -> T>(&mut self, f: F) {
6.         let mut i = 0;
7.         while i < self.len() {
8.             self[i] = f(self[i]);
```

```

9.             i += 1;
10.         }
11.     }
12. }
13. fn main(){
14.     let mut v = vec![1,2,3];
15.     v.each(|i| i * 3);
16.     assert_eq!([3, 6, 9], &v[..3]);
17. }

```

6-56 `each` Rust
 6-57 `for`

6-57 `for`

```

1. fn main() {
2.     let v = vec![1, 2, 3, 4, 5];
3.     for i in v {
4.         println!("{}", i);
5.     }
6. }

```

6-57 `for` `for` Rust
 6-58 `for`

6-58 `for`

```

1. fn main() {
2.     let v = vec![1, 2, 3, 4, 5];
3.     { // 等价于 for 循环的 scope
4.         let mut _iterator = v.into_iter();
5.         loop {
6.             match _iterator.next() {
7.                 Some(i) => {
8.                     println!("{}", i);
9.                 }
10.                None => break,
11.            }
12.        }
13.    }
14. }

```

代码6-58展示了使用 `for` 循环遍历向量的另一种方式。代码4展示了使用 `v.into_iter()` 方法将向量转换为迭代器，然后在 `loop` 循环中使用 `match` 语句调用 `next` 方法。当 `next` 返回 `None` 时，循环结束。

6.3.2 Iterator trait

在 Rust 中，`for` 循环是建立在 `Iterator` trait 之上的。代码6-59展示了 `Iterator` trait 的定义。

代码6-59 Iterator trait

```

1. trait Iterator {
2.     type Item;
3.     fn next(&mut self) -> Option<Self::Item>;
4. }

```

代码6-59展示了 `Iterator` trait 的定义。它包含一个 `Item` 类型和 `next` 方法。 `next` 方法返回一个 `Option<Self::Item>`。当 `next` 返回 `None` 时，表示迭代结束。 `Self::Item` 表示当前迭代项的类型。

代码6-60展示了 `Iterator` trait 的实现。

例6-60 Iterator trait

```
1. struct Counter {
2.     count: usize,
3. }
4. impl Iterator for Counter {
5.     type Item = usize;
6.     fn next(&mut self) -> Option<usize> {
7.         self.count += 1;
8.         if self.count < 6 {
9.             Some(self.count)
10.        } else {
11.            None
12.        }
13.    }
14. }
15. fn main() {
16.    let mut counter = Counter { count: 0 };
17.    assert_eq!(Some(1), counter.next());
18.    assert_eq!(Some(2), counter.next());
19.    assert_eq!(Some(3), counter.next());
20.    assert_eq!(Some(4), counter.next());
21.    assert_eq!(Some(5), counter.next());
22.    assert_eq!(None, counter.next());
23. }
```

例6-60 Counter Iterator の next

Counter の next は Item 型 usize の Counter の count 変数の next を Option 型

8 の next は if 文で 6 を超えたら None を返す。next は Option 型

Iterator trait の size_hint 例6-61

例6-61 Iterator trait の size_hint

```

1. pub trait Iterator {
2.     type Item;
3.     ...
4.     fn size_hint(&self) -> (usize, Option<usize>) {
5.         (0, None)
6.     }
7.     ...
8. }

```

6-61 `size_hint` 返回一个 `(usize, Option<usize>)` 元组。第一个值 `lower bound` 表示迭代器中元素的最低数量，第二个值 `upper bound` 表示元素数量的上限。如果上限未知，则返回 `None`。

6-62 实现 `size_hint`

6-62 实现 `size_hint`

```

1. fn main() {
2.     let a : [i32; 3] = [1, 2, 3];
3.     let mut iter = a.iter();
4.     assert_eq!(3, Some(3), iter.size_hint());
5.     iter.next();
6.     assert_eq!(2, Some(2), iter.size_hint());
7. }

```

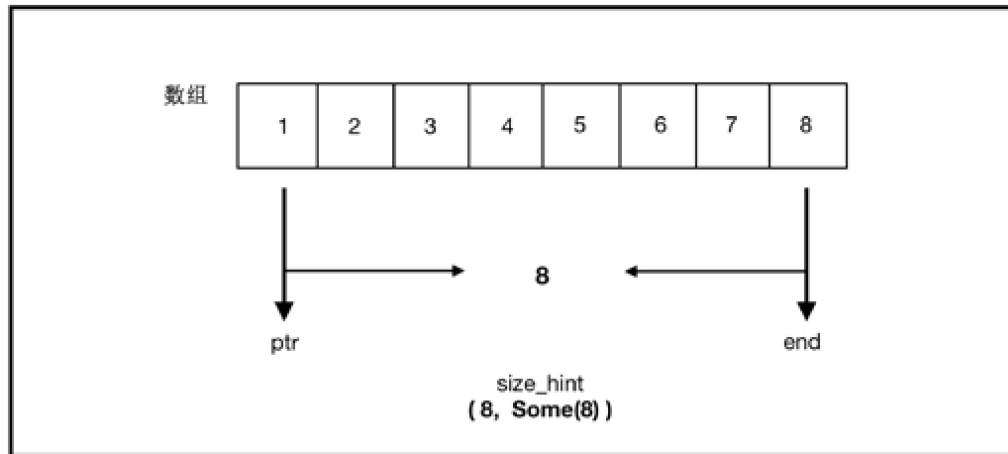
6-62 实现 `iter` 和 `next`。

0 表示 `size_hint` 返回的 `Some(3)`。

`a` 是一个 `&a [i32; 3]` 类型的引用，`&a [T]` 和 `&a mut [T]` 类型的引用。

`size_hint` 返回 `(0, Some(3))`。

6-4



6-4 `&a [T] &a mut [T]` `size_hint`

3 `a.iter` `ptr` `end` `size_hint`

`size_hint` Rust `size_hint` 6-63

6-63

```
1. fn main() {
2.     let mut message = "Hello".to_string();
3.     message.extend(&[' ', 'R', 'u', 's', 't']);
4.     assert_eq!("Hello Rust", &message);
5. }
```

6-63 `String` `message` `extend` `Extend` trait 6-64 `Extend` `String` `extend`

6-64 `Extend` `String` `extend`

```

1. pub trait Extend<A> {
2.     fn extend<T>(&mut self, iter: T)
3.     where
4.         T: IntoIterator<Item = A>;
5. }
6. ...
7. impl Extend<char> for String {
8.     fn extend<I: IntoIterator<Item = char>>(&mut self, iter: I) {
9.         let iterator = iter.into_iter();
10.        let (lower_bound, _) = iterator.size_hint();
11.        self.reserve(lower_bound);
12.        for ch in iterator {
13.            self.push(ch)
14.        }
15.    }
16. }

```

Extend trait trait extend T trait IntoIterator Item=A IntoIterator String char trait

9 String extend into_iter size_hint 10

11 reserve 6-63 5 20 4 100 reserve

12 for String push size_hint Rust trait ExactSizeIterator TrustedLen Iterator trait std iter

ExactSizeIterator len is_empty len Iterator size_hint


```
TrustedLen trait {
    Rust
    trait TrustedLen {
        size_hint
    }
    trait
```

```
ExactSizeIterator
TrustedLen
ExactSizeIterator
ExactSizeIterator
ExactSizeIterator
ExactSizeIterator
TrustedLen
```

6.3.3 IntoIterator trait

```
Iterator trait {
    for into_iter
    IntoIterator trait
```

```
3 From Into trait from
into trait Rust
trait
```

```
Rust FromIterator IntoIterator trait
FromIterator IntoIterator
FromIterator 6.3.5 IntoIterator
```

6-65 IntoIterator

6-65 IntoIterator

```
1. pub trait IntoIterator {
2.     type Item;
3.     type IntoIter: Iterator<Item=Self::Item>;
4.     fn into_iter(self) -> Self::IntoIter;
5. }
```

```
6-65 into_iter trait into_iter
self Self IntoIter Self
IntoIter trait Iterator Item=Self Item
Iterator
```

```
Vec T IntoIterator into_iter
6-66 Vec T IntoIterator
```

6-66 Vec<T> IntoIterator

```
1. impl<T> IntoIterator for Vec<T> {
2.     type Item = T;
3.     type IntoIter = IntoIter<T>;
4.     fn into_iter(mut self) -> IntoIter<T> {
5.         unsafe {
6.             ...
7.             IntoIter {
8.                 buf: Shared::new_unchecked(begin),
9.                 cap,
10.                ptr: begin,
11.                end,
12.            }
13.        }
14.    }
15. }
```

6-66 实现了 Vec<T> 的 IntoIterator trait，std::vec::IntoIter 是 Vec<T> 的迭代器。

- **Buf** Vec<T> 的 begin 指针指向的 Shared 缓冲区。

- **Cap** 缓冲区的容量。

- **Ptr** begin 指针。

- **End** Vec<T> 的 len 减去 begin 的 offset。

IntoIter 实现了 Iterator trait 的 next、size_hint、count 方法。

Vec<T> 的 IntoIterator 的 into_iter 方法返回 Vec<T> 的 IntoIter 迭代器。IntoIter 实现了 Iterator trait 的 next、size_hint、count 方法。IntoIter 的 Vec<T> 的 into_iter 方法返回 6-5。

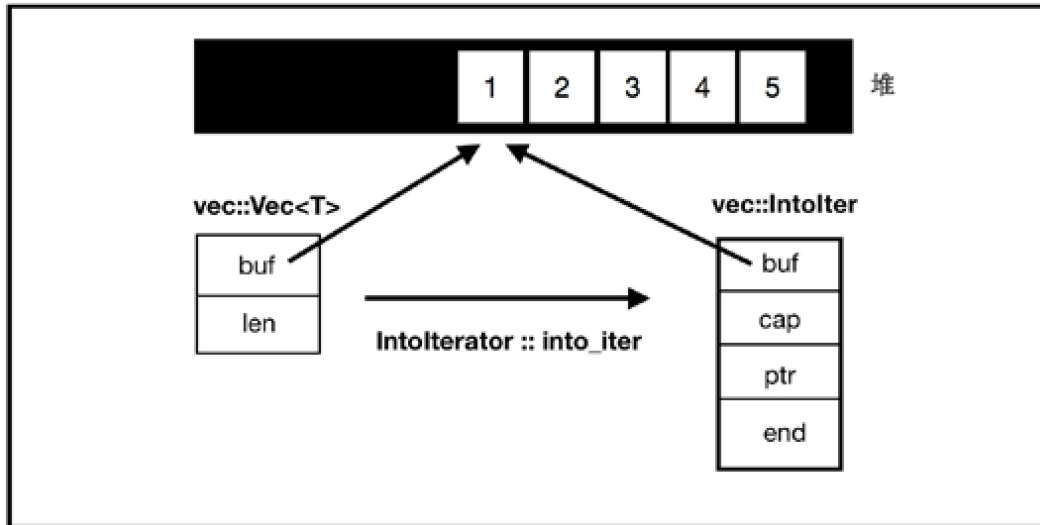


图6-5 Vec<T>的迭代器

而 `IntoIter` 接口定义了遍历集合的接口。在 Rust 中，我们使用 `Iter` 和 `IterMut` 接口来遍历集合。

- **IntoIter** 接口定义了 `self`
- **Iter** 接口定义了 `&self`
- **IterMut** 接口定义了 `&mut self`

`Iter` 和 `IterMut` 接口都实现了 `slice` 方法，如图6-67所示。slice 方法返回一个切片。

图6-67 slice 方法

```
1. fn main() {
2.     let arr = [1, 2, 3, 4, 5];
3.     for i in arr.iter() {
4.         println!("{:?}", i);
5.     }
6.     println!("{:?}", arr);
7. }
```

图6-67 slice 方法返回一个切片。for 循环遍历切片。切片是 `[T]` 类型。在 `IntoIterator` 接口中，`&a [T]` 和 `&a mut [T]` 都是 `IntoIterator` 接口的实现。它们分别实现了 `iter` 和 `iter_mut` 方法。

for &arr iter iter
iter_mut slice Iter IterMut

6-68 Iter

6-68 Iter

```
1. pub struct Iter<'a, T: 'a> {  
2.     ptr: *const T,  
3.     end: *const T,  
4.     _marker: marker::PhantomData<&'a T>,  
5. }
```

Iter ptr end *const T
_marker a
PhantomData 13

Iter 6-69
IterMut

6-69 IterMut

```
1. pub struct IterMut<'a, T: 'a> {  
2.     ptr: *mut T,  
3.     end: *mut T,  
4.     _marker: marker::PhantomData<&'a mut T>,  
5. }
```

IterMut ptr end
6-70 iter_mut for
slice

6-70

```
1. fn main() {  
2.     let mut arr = [1, 2, 3, 4, 5];  
3.     for i in arr.iter_mut() {  
4.         *i += 1;  
5.     }  
6.     println!("{:?}", arr); // [2, 3, 4, 5, 6]  
7. }
```

Rust 提供了 `IntoIter`、`Iter`、`IterMut` 等 trait，`String`、`HashMap` 实现了 **`Drain`** trait，`HashMap` 提供了 `drain` 方法，`Rust` 提供了 `into_iter`、`iter`、`iter_mut` 等方法，`Rust` 提供了 `slice`、`vec`、`slice::Iter`、`Vec::Iter`、`HashMap::Iter` 等 trait，`HashMap` 提供了 `into_iter`、`iter`、`iter_mut` 等方法，`HashMap` 提供了 `drain` 方法。

6.3.4 迭代器

迭代器（Iterator）是 Rust 中一个非常重要的概念，它提供了一种遍历集合（如数组、向量、字符串、哈希表等）的方法。迭代器接口（Iterator Trait）定义了 `next` 方法，该方法返回一个 `Option` 类型的值，表示下一个元素是否存在。如果存在，则返回 `Some` 包裹的元素；如果不存在，则返回 `None`。

在 Rust 中，`Iterator` trait 定义在 `std::iter` 模块中。许多集合类型都实现了这个 trait，例如 `Vec`、`String`、`HashMap` 等。此外，Rust 还提供了 `IntoIterator` trait，用于将某些类型转换为 `Iterator`。

Map

Map 是 Rust 中一个非常常用的迭代器方法，它用于对集合中的每个元素进行映射（transform）操作。

6-71 map

```
1. fn main() {
2.     let a = [1, 2, 3];
3.     let mut iter = a.into_iter().map(|x| 2 * x);
4.     assert_eq!(iter.next(), Some(2));
5.     assert_eq!(iter.next(), Some(4));
6.     assert_eq!(iter.next(), Some(6));
7.     assert_eq!(iter.next(), None);
8. }
```

在 6-71 中，我们使用 `into_iter` 方法将数组 `a` 转换为一个 `IntoIterator` 类型的值，然后使用 `map` 方法对其进行映射操作。映射操作的结果是一个 `Iterator` 类型的值，我们可以使用 `next` 方法来遍历这个迭代器。

map 6-72 std
iter Iterator map

6-72 map

```
1. pub trait Iterator {  
2.     type Item;  
3.     ...  
4.     fn map<B, F>(self, f: F) -> Map<Self, F>  
5.     where  
6.         Self: Sized,  
7.         F: FnMut(Self::Item) -> B,  
8.     {  
9.         Map { iter: self, f: f }  
10.    }  
11. }
```

map Iterator trait self Iterator
f FnMut trait FnMut Self Item B
Self Item Iterator Item
Map Self F Self Sized Self
Map

6-73 Map

6-73 Map

```

1. #[must_use="iterator adaptors are lazy ....."]
2. #[derive(Clone)]
3. pub struct Map<I, F> {
4.     iter: I,
5.     f: F,
6. }
7. impl<B, I: Iterator, F> Iterator for Map<I, F>
8.     where F: FnMut(I::Item) -> B
9.     {
10.     type Item = B;
11.     fn next(&mut self) -> Option<B> {
12.         self.iter.next().map(&mut self.f)
13.     }
14.     fn size_hint(&self) -> (usize, Option<usize>) {
15.         self.iter.size_hint()
16.     }
17.     ...
18. }

```

Map の構造は、**iter** と **f** の 2 つのフィールドからなる。また、Iterator trait の Map 実装では、next と size_hint の 2 つのメソッドを実装する。12 行の map は、next の戻り値 Option<T> の map を利用して、12 行の map を利用して Map の next を Option<T> の形式で返す。

6-73 Map の構造と 6-71 Map の実装を比較すると、[must_use=.....] の注釈が追加されている。これは、"iterator adaptors are lazy" という意味で、Rust の must_use 属性を付与している。

Map の構造と 6-71 Map の実装を比較すると、6-6 のように

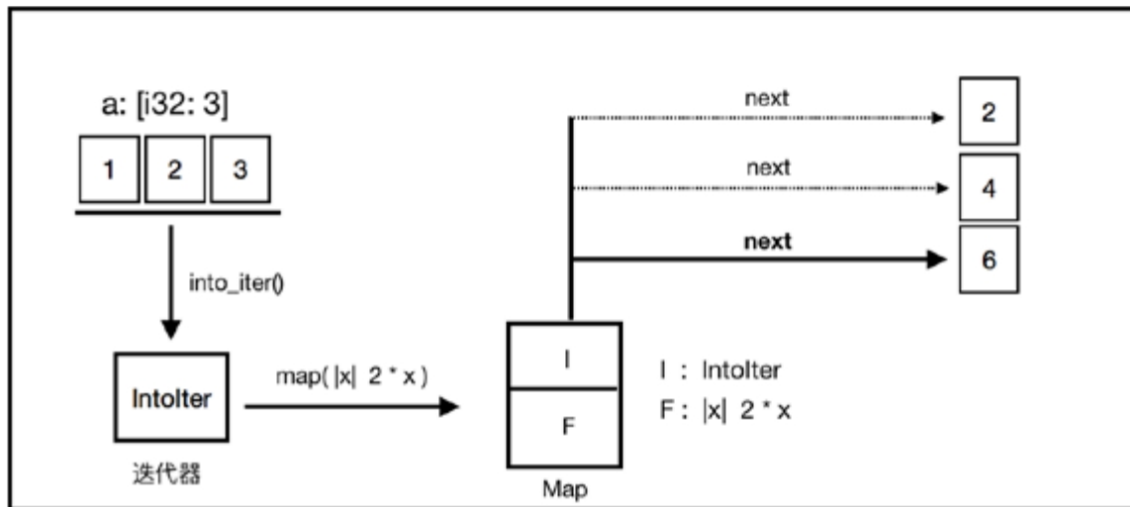


图6-6 图6-71的map操作

图6-6 图6-71 展示了 Rust 中的 `Iterator` trait 和 `map` 操作。图中显示了一个数组 `a: [i32; 3]` 包含元素 `1, 2, 3`。通过调用 `into_iter()`，该数组被转换为一个 `Intolter`（迭代器）。然后，使用 `map(|x| 2 * x)` 对该迭代器进行映射操作，生成一个新的 `Map` 结构。该结构包含一个 `Intolter`（标记为 `I`）和一个函数 `F: |x| 2 * x`。通过调用 `next` 方法，可以依次获取映射后的值 `2, 4, 6`。

在 Rust 中，`Map` 是 `std::iter` 模块中的一个 trait，用于对迭代器进行映射操作。它允许我们对集合中的每个元素应用一个函数，并将结果收集到一个新的集合中。图中展示了 `Intolter` 和 `Map` 的结构，以及 `next` 方法的调用过程。

- **Map** 对集合中的每个元素应用一个函数，并将结果收集到一个新的集合中。
- **Chain** 将两个集合连接起来，按顺序遍历。
- **Cloned** 对集合中的每个元素进行克隆，并收集到一个新的集合中。
- **Cycle** 对集合中的每个元素进行无限循环遍历。
- **Enumerate** 对集合中的每个元素进行遍历，并返回元素的索引和值。例如：

```

        for (i, val) in enumerate(<iter>) {
            // ...
        }
    
```
- **Filter** 对集合中的每个元素应用一个谓词（predicate），如果谓词返回 `true`，则将元素收集到一个新的集合中。
- **FlatMap** 对集合中的每个元素应用一个函数，该函数返回一个集合，然后将所有结果集合扁平化（flatten）成一个集合。
- **FilterMap** 对集合中的每个元素应用一个函数，该函数返回一个集合，然后将所有结果集合扁平化（flatten）成一个集合。
- **Fuse** 对集合中的每个元素进行遍历，一旦遇到 `None`，则停止遍历。
- **Rev** 对集合中的元素进行反向遍历。

6-74

6-74

```
1.fn main() {
2.    let arr1 = [1, 2, 3, 4, 5];
3.    let c1 = arr1.iter().map(|x| 2 * x).collect::<Vec<i32>>();
4.    assert_eq!(&c1[..], [2, 4, 6, 8, 10]);
5.    let arr2 = ["1", "2", "3", "h"];
6.    let c2 = arr2.iter().filter_map(|x| x.parse().ok())
7.        .collect::<Vec<i32>>();
8.    assert_eq!(&c2[..], [1,2,3]);
9.    let arr3 = ['a', 'b', 'c'];
10.   for (idx, val) in arr3.iter().enumerate() {
11.       println!("idx: {:?}, val: {}", idx, val.to_uppercase());
12.   }
13. }
```

6-74 2 4 map Map collect 3 Vec i32

5 8 filter_map FilterMap collect 7 Vec i32

9 12 enumerate Enumerate for next 10 for idx val

Rev rev 6-75

6-75 reü

```
1. fn main() {
2.     let a = [1, 2, 3];
3.     let mut iter = a.iter().rev();
4.     assert_eq!(iter.next(), Some(&3));
5.     assert_eq!(iter.next(), Some(&2));
6.     assert_eq!(iter.next(), Some(&1));
7.     assert_eq!(iter.next(), None);
8. }
```

6-75 rev next
“” 6-76 rev

6-76 reü

```
1. pub trait Iterator {  
2.     type Item;  
3.     fn rev(self) -> Rev<Self>  
4.     where Self: Sized + DoubleEndedIterator,  
5.     {  
6.         Rev { iter: self }  
7.     }  
8. }
```

rev Rev Self
trait **DoubleEndedIterator** trait trait

6-77 Rev

6-77 Reü

```
1. pub struct Rev<T> {  
2.     iter: T,  
3. }  
4. impl<I> Iterator for Rev<I>  
5.     where I: DoubleEndedIterator,  
6.     {  
7.         type Item = <I as Iterator>::Item;  
8.         fn next(&mut self) -> Option<<I as Iterator>::Item> {  
9.             self.iter.next_back()  
10.        }  
11. }
```

Rev iter Iterator
DoubleEndedIterator Item Iterator

next Rec next_back
next_back DoubleEndedIterator 6-78

6-78 DoubleEndedIterator

```

1. pub trait DoubleEndedIterator: Iterator {
2.     fn next_back(&mut self) -> Option<Self::Item>;
3. }

```

6-78 DoubleEndedIterator trait Iterator next_back next 6-79 next_back

6-79 next_back

```

1. fn main() {
2.     let numbers = vec![1, 2, 3, 4, 5, 6];
3.     let mut iter = numbers.into_iter();
4.     assert_eq!(Some(1), iter.next());
5.     assert_eq!(Some(6), iter.next_back());
6.     assert_eq!(Some(5), iter.next_back());
7.     assert_eq!(Some(2), iter.next());
8.     assert_eq!(Some(3), iter.next());
9.     assert_eq!(Some(4), iter.next());
10.    assert_eq!(None, iter.next());
11.    assert_eq!(None, iter.next_back());
12. }

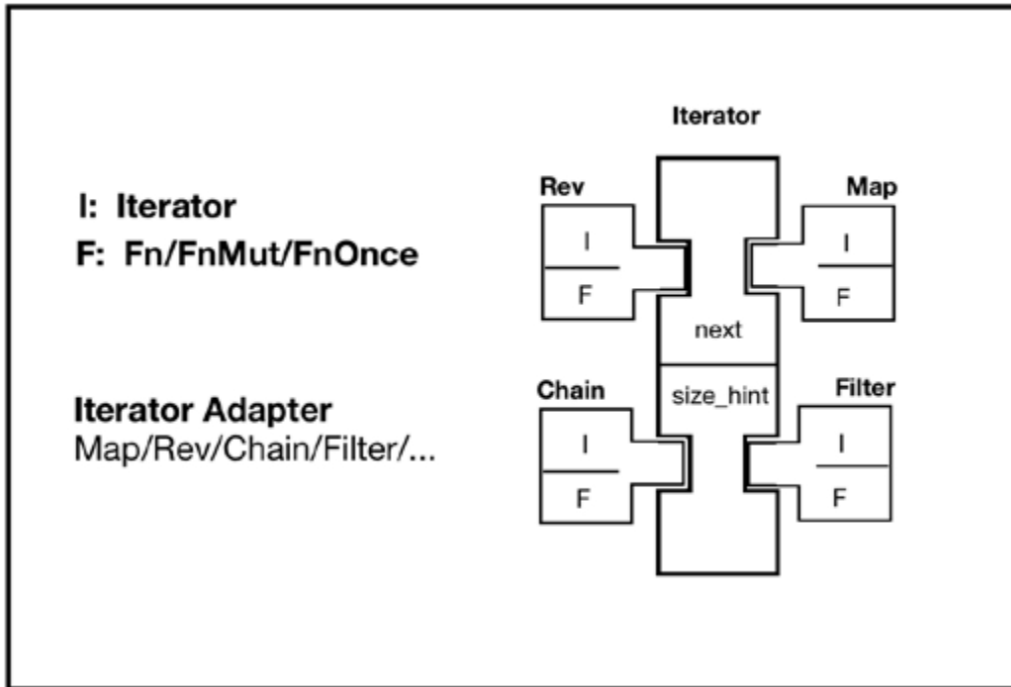
```

6-79 4 next Some 1

5 6 next_back Some 6 Some 5
 7 9 next Some 2
 Some 3 Some 4 next_back “ ”
 next next next_back 10
 11 None

Rev next next_back
 DoubleEndedIterator next_back
 DoubleEndedIterator Iterator rev

Rust
 6-7



6-7

6.3.5

Rust 的 `Iterator` trait 定义了 `next` 方法，用于遍历集合。在 Rust 中，`for` 循环和 `next` 方法是紧密相关的。

在 Rust 中，`for` 循环和 `next` 方法是紧密相关的。在 Rust 中，`for` 循环和 `next` 方法是紧密相关的。

· **any** 6-42 任何 `any` 方法

· **fold** 6-74 `fold` 方法

· **collect** 6-74 `collect` 方法

any **fold**

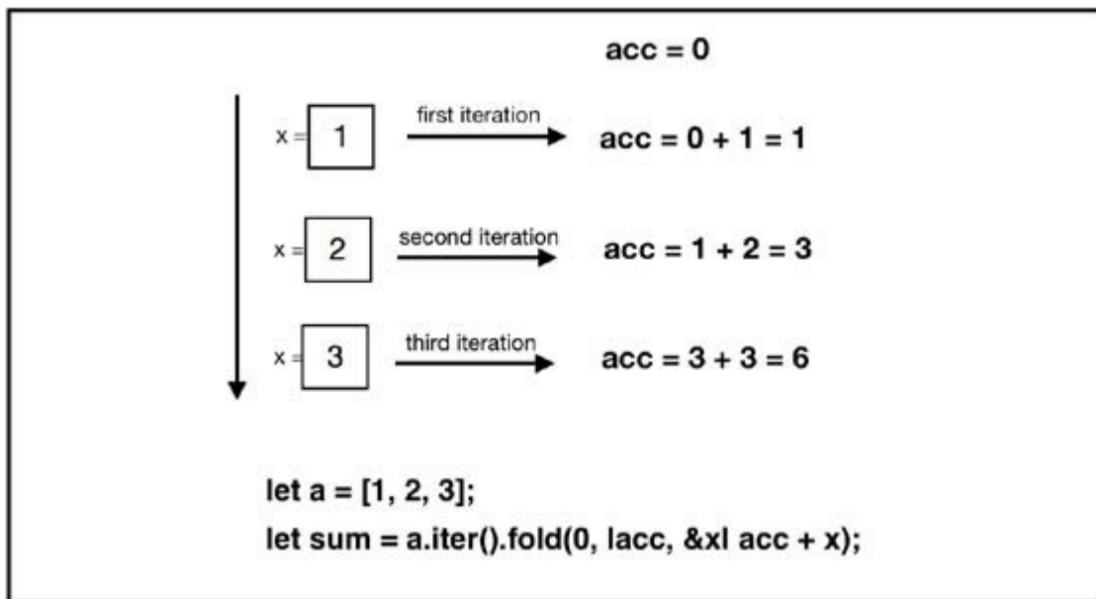
6-80 any fold

6-80 any fold

```
1. fn main() {  
2.     let a = [1, 2, 3];  
3.     assert_eq!(a.iter().any(|&x| x != 2), true);  
4.     let sum = a.iter().fold(0, |acc, x| acc + x);  
5.     assert_eq!(sum, 6);  
6. }
```

6-80 3 any a 2 true

4 fold a 6-8 fold



6-8 fold

6-80 any fold

6-81 any fold

6-81 any fold

```

1. pub trait Iterator {
2.     type Item;
3.     ...
4.     fn any<F>(&mut self, mut f: F) -> bool
5.     where Self: Sized,
6.           F: FnMut(Self::Item) -> bool,
7.     {
8.         for x in self {
9.             if f(x) {
10.                return true;
11.            }
12.        }
13.        false
14.    }
15.    ...
16.    fn fold<B, F>(self, init: B, mut f: F) -> B
17.    where Self: Sized,
18.          F: FnMut(B, Self::Item) -> B,
19.    {
20.        let mut accum = init;
21.        for x in self {
22.            accum = f(accum, x);
23.        }
24.        accum
25.    }
26. }

```

any fold for for
 return break continue
 LLVM

6-80 3 iter Iter
 next Option &[T] Option &mut [T] for
 next for next
 Option &[T] Option &mut [T] &[T] &mut [T]

6-81 any for x
6-80 3 any 6-82

6-82 any

```
1. fn main() {
2.     let arr = [1, 2, 3];
3.     let result1 = arr.iter().any(|&x| x != 2);
4.     let result2 = arr.iter().any(|x| *x != 2);
5.     // error:
6.     // the trait bound `{integer}: std::cmp::PartialEq<{integer}>` is
    not satisfied
7.     // let result2 = arr.iter().any(|x| x != 2);
8.     assert_eq!(result1, true);
          9.     assert_eq!(result2, true);
          10. }
```

6-82 3 4 any &x x
&x any x
x 7
x

fold 6-83

6-83 fold

```
1. fn main() {
2.     let arr = vec![1, 2, 3];
3.     let sum1 = arr.iter().fold(0, |acc, x| acc + x);
4.     let sum2 = arr.iter().fold(0, |acc, x| acc + *x);
5.     let sum3 = arr.iter().fold(0, |acc, &x| acc + x);
6.     let sum4 = arr.into_iter().fold(0, |acc, x| acc + x);
7.     assert_eq!(sum1, 6);
8.     assert_eq!(sum2, 6);
9.     assert_eq!(sum3, 6);
10.    assert_eq!(sum4, 6);
11. }
```

6-83 3 4 5 iter arr
Iter 6 into_iter IntoIter arr

Iter for fold for
6-83 3 4 5

IntoIter next Option T for
6-83 6 fold for
6 x &x x*x

Rust any fold all
for_each position std iter Iterator
collect

collect

collect “”
collect Vec i32
Vec i32 turbofish
collect HashMap i32 i32 6-84 collect

6-84 collect

```
1. pub trait Iterator {  
2.     type Item;  
3.     ...  
4.     fn collect<B: FromIterator<Self::Item>>(self) -> B where Self:  
       Sized {  
5.         FromIterator::from_iter(self)  
6.     }  
7. }
```

collect FromIterator from_iter
FromIterator IntoIterator trait
6-85 FromIterator

6-85 FromIterator


```

1. pub trait FromIterator<A>: Sized {
2.     fn from_iter<T: IntoIterator<Item=A>>(iter: T) -> Self;
3. }

```

trait from_iter trait IntoIterator<Item=A> IntoIterator collect 6-86 MyVec FromIterator trait

6-86 MyVec FromIterator

```

1. use std::iter::FromIterator;
2. #[derive(Debug)]
3. struct MyVec(Vec<i32>);
4. impl MyVec {
5.     fn new() -> MyVec {
6.         MyVec(Vec::new())
7.     }
8.     fn add(&mut self, elem: i32) {
9.         self.0.push(elem);
10.    }
11. }
12. impl FromIterator<i32> for MyVec {
13.     fn from_iter<I: IntoIterator<Item = i32>>(iter: I) -> Self {
14.         let mut c = MyVec::new();
15.         for i in iter {
16.             c.add(i);
17.         }
18.         c
19.     }
20. }
21. fn main() {
22.     let iter = (0..5).into_iter();
23.     let c = MyVec::from_iter(iter);
24.     assert_eq!(c.0, vec![0, 1, 2, 3, 4]);
25.     let iter = (0..5).into_iter();
26.     let c: MyVec = iter.collect();
27.     assert_eq!(c.0, vec![0, 1, 2, 3, 4]);
28.     let iter = (0..5).into_iter();
29.         let c = iter.collect::<MyVec>();
30.         assert_eq!(c.0, vec![0, 1, 2, 3, 4]);
31. }

```

6-86 Vec<i32> MyVec
 FromIterator main collect
 MyVec

MyVec::from_iter::collect
::

6.3.6

Rust
Rust Rust

Step::I 6-87

6-87 Step::I

1. `#[derive(Clone, Debug)]`
2. `#[must_use = "iterator adaptors are lazy and do nothing unless consumed"]`
3. `pub struct Step<I> {`
4. `iter: I,`
5. `skip: usize,`
6. `}`

6-87 Step::I iter
skip Iterator 6-88

6-88 Step Iterator

1. `impl<I> Iterator for Step<I>`
2. `where I: Iterator,`
3. `{`
4. `type Item = I::Item;`
5. `fn next(&mut self) -> Option<I::Item> {`
6. `let elt = self.iter.next();`
7. `if self.skip > 0 {`
8. `self.iter.nth(self.skip - 1);`
9. `}`
10. `elt`
11. `}`
12. `}`

6-88 Step Iterator next
Item I Item

next size_hint Iterator trait next
next next Step
skip skip 2 next
8 nth n

step Step 6-89

6-89 step Step

```
1. pub fn step<I>(iter: I, step: usize) -> Step<I>
2.   where I: Iterator,
3.   {
4.       assert!(step != 0);
5.       Step {
6.           iter: iter,
7.           skip: step - 1,
8.       }
9.   }
```

6-89 step
Step

step
6-90

6-90 step

```
1. pub trait IterExt: Iterator {
2.   fn step(self, n: usize) -> Step<Self>
3.   where Self: Sized,
4.   {
5.       step(self, n)
6.   }
7. }
8. impl<T: ?Sized> IterExt for T where T: Iterator {}
```

6-90 Iterator trait
IterExt step step Step

8 impl Iterator for T::IterExt
6-91

6-91 Step

```
1. fn main() {  
2.     let arr = [1,2,3,4,5,6];  
3.     let sum = arr.iter().step(2).fold(0, |acc, x| acc + x);  
4.     assert_eq!(9, sum); // [1, 3, 5]  
5. }
```

6-91 arr.iter().step(2).fold(0, |acc, x| acc + x)
2 [1 3 5] fold 9

Rust crate
Itertools 6-92 Itertools Positions

6-92 Itertools Positions

```
1. #[must_use = "iterator adaptors are lazy and do nothing unless
```

```

consumed"]
2.  #[derive(Debug)]
3.  pub struct Positions<I, F> {
4.      iter: I,
5.      f: F,
6.      count: usize,
7.  }
8.  pub fn positions<I, F>(iter: I, f: F) -> Positions<I, F>
9.      where I: Iterator,
10.     F: FnMut(I::Item) -> bool,
11.     {
12.     Positions {
13.         iter: iter,
14.         f: f,
15.         count: 0
16.     }
17. }
18. impl<I, F> Iterator for Positions<I, F>
19.     where I: Iterator,
20.     F: FnMut(I::Item) -> bool,
21.     {
22.     type Item = usize;
23.     fn next(&mut self) -> Option<Self::Item> {
24.         while let Some(v) = self.iter.next() {
25.             let i = self.count;
26.             self.count = i + 1;
27.             if (self.f)(v) {
28.                 return Some(i);
29.             }
30.         }
31.         None
32.     }
33.     fn size_hint(&self) -> (usize, Option<usize>) {
34.         (0, self.iter.size_hint().1)
35.     }
36. }
37. impl<I, F> DoubleEndedIterator for Positions<I, F>
38.     where I: DoubleEndedIterator + ExactSizeIterator,
39.     F: FnMut(I::Item) -> bool,
40.     {
41.     fn next_back(&mut self) -> Option<Self::Item> {
42.         while let Some(v) = self.iter.next_back() {
43.             if (self.f)(v) {
44.                 return Some(self.count + self.iter.len())
45.             }
46.         }
47.         None
48.     }
49. }
50. pub trait Itertools: Iterator {
51.     fn positions<P>(self, predicate: P) -> Positions<Self, P>

```

```

52.     where Self: Sized,
53.     P: FnMut(Self::Item) -> bool,
54.     {
55.         positions(self, predicate)
56.     }
57. }
58. impl<T: ?Sized> Itertools for T where T: Iterator {}
59. fn main() {
60.     let data = vec![1, 2, 3, 3, 4, 6, 7, 9];
61.     let r = data.iter().positions(|v| v % 3 == 0);
62.     let rev_r = data.iter().positions(|v| v % 3 == 0).rev();
63.     for i in r { println!("{:?}", i); } // OUTPUT: 2 3 5 7
64.     for i in rev_r { println!("{:?}", i); } // OUTPUT: 7 5 3 2
65. }

```

6-92 Itertools Positions

main

3 7 Positions iter f

count

8 17 positions Positions

18 36 Positions Iterator next size_hint

next count

37 49 Positions DoubleEndIterator

ExactSizeIterator

50 57 Itertools Iterator trait

Iterator trait positions

main 61 62 positions Positions

for

Positions Itertools

crates.io

6.4

Rust Rust fn Rust

Rust trait Fn FnMut FnOnce Rust trait Rust 2018 impl Trait trait impl Trait Trait Fn FnMut FnOnce trait

Rust Rust trait Rust for for next

Rust collect fold return break continue for std iter

Rust Rust trait Rust

第7章 木结构

木结构工程

木结构工程是指以木材为主要材料，通过榫卯、胶合等方式连接而成的结构体系。本章主要介绍木结构的基本构造、设计要求和施工方法。木结构具有取材广泛、施工简便、抗震性能好等优点，广泛应用于古建筑、现代轻木结构房屋等。

本章主要介绍木结构的基本构造、设计要求和施工方法。木结构具有取材广泛、施工简便、抗震性能好等优点，广泛应用于古建筑、现代轻木结构房屋等。

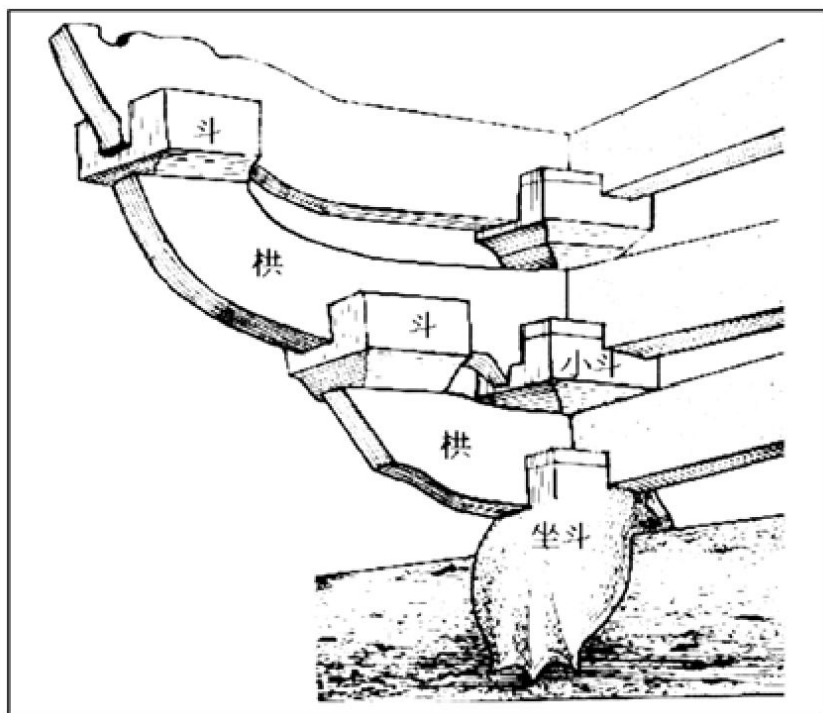


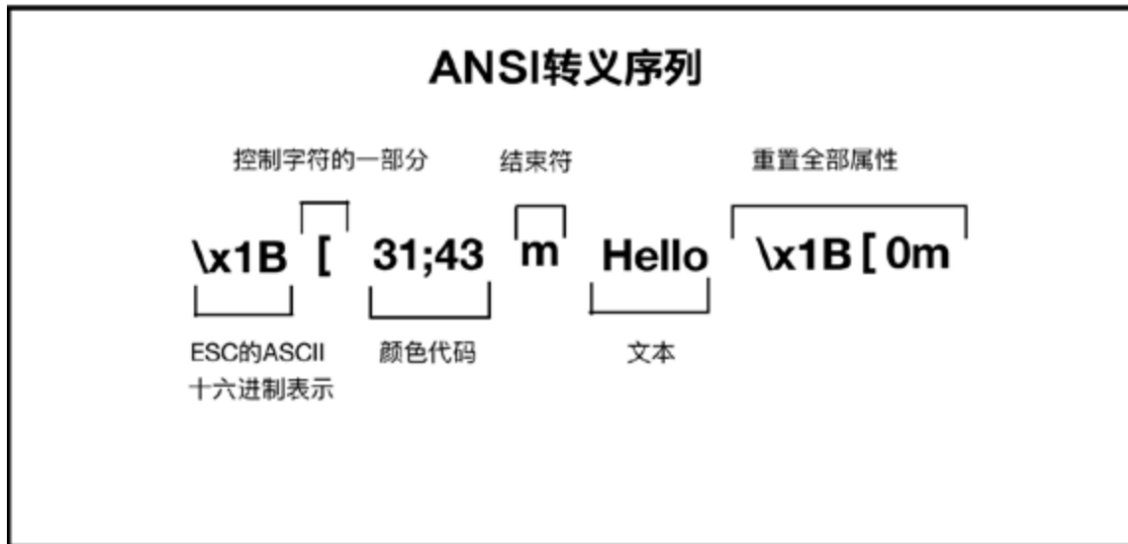
图7-1 木结构基本构造

木结构工程是指以木材为主要材料，通过榫卯、胶合等方式连接而成的结构体系。本章主要介绍木结构的基本构造、设计要求和施工方法。木结构具有取材广泛、施工简便、抗震性能好等优点，广泛应用于古建筑、现代轻木结构房屋等。

本章主要介绍木结构的基本构造、设计要求和施工方法。木结构具有取材广泛、施工简便、抗震性能好等优点，广泛应用于古建筑、现代轻木结构房屋等。

通过echo命令Linux终端输出Hello

\x1B[31;43 m 通过ANSI转义序列输出**31;43** 通过31;43mHello
\x1B[0m 通过ANSI转义序列7-2 ESC 通过ANSI转义序列



7-2 ANSI转义序列

通过Hello通过ANSI转义序列输出ANSI转义序列

- 1.通过ANSI转义序列输出
- 2.通过ANSI转义序列输出red on yellow
- 3.通过ANSI转义序列输出red on yellow & a str
red on yellow
- 4.通过ANSI转义序列输出ANSI转义序列
- 5.通过

通过ANSI转义序列输出 **color.rs** 通过ANSI转义序列输出Rust
.rs 通过

1通过ANSI转义序列输出ColoredString通过ANSI转义序列输出
通过ANSI转义序列7-4

通过7-4 **color.rs** 通过**ColoredString**

```

1. struct ColoredString {
2.     input: String,
3.     fgcolor: String,
4.     bgcolor: String,
5. }

```

ColoredString String input fgcolor bgcolor

2 3 &a str trait Colorize trait 7-5 Rust trait C++ Java

7-5 color.rs Colorize trait

```

1. trait Colorize {
2.     const FG_RED : &'static str = "31";
3.     const BG_YELLOW : &'static str = "43";
4.     fn red(self) -> ColoredString;
5.     fn on_yellow(self) -> ColoredString;
6. }

```

7-5 Colorize trait trait FG_RED BG_YELLOW Rust 2018 trait 7-5 trait

trait red on_yellow self ColoredString trait self self Self trait Rust receiver.message receiver

impl ColoredString {
 fn new(a str: String) -> ColoredString {
 let red = String::from("31");
 let fgcolor = red;
 let bgcolor = String::from("43");
 let on_yellow = bgcolor;
 ColoredString {
 input: a str,
 fgcolor: fgcolor,
 bgcolor: bgcolor,
 on_yellow: on_yellow,
 }
 }
}

```
ColoredString {  
    input: String::new(),  
    fgcolor: String::from("31"),  
    bgcolor: String::new(),  
}
```

impl ColoredString {
 fn new(a str: String) -> ColoredString {
 let red = String::from("31");
 let fgcolor = red;
 let bgcolor = String::from("43");
 let on_yellow = bgcolor;
 ColoredString {
 input: a str,
 fgcolor: fgcolor,
 bgcolor: bgcolor,
 on_yellow: on_yellow,
 }
 }
}

```
ColoredString { fgcolor: String::from("31"), ..self }  
ColoredString { bgcolor: String::from("43"), ..self }
```

impl ColoredString {
 fn new(a str: String) -> ColoredString {
 let red = String::from("31");
 let fgcolor = red;
 let bgcolor = String::from("43");
 let on_yellow = bgcolor;
 ColoredString {
 input: a str,
 fgcolor: fgcolor,
 bgcolor: bgcolor,
 on_yellow: on_yellow,
 }
 }
}

7-6 color.rs ColoredString Default

```
1. impl Default for ColoredString {  
2.     fn default() -> Self {  
3.         ColoredString {  
4.             input: String::default(),  
5.             fgcolor: String::default(),  
6.             bgcolor: String::default(),  
7.         }  
8.     }  
9. }
```

impl Default for ColoredString {
 fn default() -> ColoredString {
 let red = String::from("31");
 let fgcolor = red;
 let bgcolor = String::from("43");
 let on_yellow = bgcolor;
 ColoredString {
 input: String::default(),
 fgcolor: fgcolor,
 bgcolor: bgcolor,
 on_yellow: on_yellow,
 }
 }
}

7-7 color.rs ColoredString & a str Colorize

```
1. impl<'a> Colorize for ColoredString {
2.     fn red(self) -> ColoredString {
3.         ColoredString{
4.             fgcolor: String::from(ColoredString::FG_RED), ..self
5.         }
6.     }
7.     fn on_yellow(self) -> ColoredString {
8.         ColoredString {
9.             bgcolor: String::from(ColoredString::BG_YELLOW), ..self
10.        }
11.    }
12. }
13. impl<'a> Colorize for &'a str {
14.     fn red(self) -> ColoredString {
15.         ColoredString {
16.             fgcolor: String::from(ColoredString::FG_RED),
17.             input: String::from(self),
18.             ..ColoredString::default()
19.         }
20.     }
21.     fn on_yellow(self) -> ColoredString {
22.         ColoredString {
23.             bgcolor: String::from(ColoredString::BG_YELLOW),
24.             input: String::from(self),
25.             ..ColoredString::default()
26.         }
27.     }
28. }
```

ColorizeHello.redHello.red
on_yellowColoredString

impl Colorize for ColoredString {
 ColoredString Colorize

7-749..self1825
 ..ColoredString default red
 on_yellow ColoredString
 ColoredString self

ColoredString Colorize
 ColoredString FG_RED
 ColoredString BG_YELLOW ANSI
 ColoredString compute_style 7-8

7-8 color.rs ColoredString compute_style

```
1. impl ColoredString{  
2.     fn compute_style(&self) -> String {  
3.         let mut res = String::from("\x1B[");  
4.         let mut has_wrote = false;  
5.         if !self.bgcolor.is_empty() {  
6.             res.push_str(&self.bgcolor);  
7.             has_wrote = true;  
8.         }  
9.         if !self.fgcolor.is_empty() {  
10.            if has_wrote { res.push(';'); }  
11.            res.push_str(&self.fgcolor);  
12.        }  
13.        res.push('m');  
14.        res  
15.    }  
16. }
```

7-8 ColoredString compute_style ANSI
 \x1B[43;31m \x1B ANSI
 43;31 43;31 m

ANSI

7-83StringresANSI

4boolhas_wrotebgcolor
58Stringis_empty
bgcolor
push_str res push_str &str
&self.bgcolor &String &str
has_wrotetrue

912fgcolorfgcolorres
has_wrote res
ANSI
433131
43

1314mresANSI

\x1B[4331mHello\x1B[0m
coloredstringDisplay7-9

7-9color.rs ColoredStringDisplay

```
1. use std::fmt;
2. impl fmt::Display for ColoredString {
3.     fn fmt(&self, f: &mut fmt::Formatter) -> fmt::Result {
4.         let mut input = &self.input.clone();
5.         try!(f.write_str(&self.compute_style()));
6.         try!(f.write_str(input));
7.         try!(f.write_str("\x1B[0m"));
8.         Ok(())
9.     }
10. }
```

Display stdfmttraitDebugDebug
trait{ } Debug

Debug 属性 `#[derive(Debug)]` 使用 `Display` 属性 `{}` 使用 `Debug` 属性

7-9 使用 `Display` 属性 `fmt` 和 `&self` 属性
`fmt` 属性 `Formatter` 属性
`write_str` 属性 `fmt` 属性 `Result` 属性
`Option` 属性

4 使用 `input` 属性 `ColoredString` 属性 `input` 属性

5 使用 `write_str` 属性 `ANSI` 属性
`try` 属性 `Rust` 属性
`Err` 属性 `Ok` 属性

`main` 属性 7-10

7-10 `color.rs` `main`

```
1. fn main() {  
2.     let hi = "Hello".red().on_yellow();  
3.     println!("{}", hi);  
4.     let hi = "Hello".on_yellow();  
5.     println!("{}", hi);  
6.     let hi = "Hello".red();  
7.     println!("{}", hi);  
8.     let hi = "Hello".on_yellow().red();  
9.     println!("{}", hi);  
10. }
```

7-10 `main` 属性 `red` 属性 `on_yellow` 属性
`ANSI` 属性 `rustc` 属性 `color.rs` 属性

```
$ rustc color.rs  
$ ./color
```

`Rust` 属性 `trait` 属性

7-13
7-13

7-13

1. class Color{}
2. class Red: Color{}
3. class Yellow: Color{}
4. class Blue: Color{}

7-13 Ruby Python Java C++ Color Red Yellow Blue Color Rust

color.rs

-
- if fgcolor bgcolor
-

color.rs

7-12 Color Color ANSI 7-14

7-14 Color ANSI

```

1. impl Color {
2.     fn to_fg_str(&self) -> &str {
3.         match *self {
4.             Color::Red => "31",
5.             Color::Yellow => "33",
6.             Color::Blue => "34",
7.         }
8.     }
9.     fn to_bg_str(&self) -> &str {
10.        match *self {
11.            Color::Red => "41",
12.            Color::Yellow => "43",
13.            Color::Blue => "44",
14.        }
15.    }
16. }

```

7-14 impl Color to_fg_str to_bg_str
 ANSI match Color
 Rust 3 10
 *self self

ColoredString fgcolor bgcolor
 7-15

7-15 ColoredString fgcolor bgcolor

```

1. #[derive(Clone, Debug, PartialEq, Eq)]
2. struct ColoredString {
3.     input: String,
4.     fgcolor: Option<Color>,
5.     bgcolor: Option<Color>,
6. }
7. impl Default for ColoredString {
8.     fn default() -> Self {
9.         ColoredString {
10.             input: String::default(),

```

```

11.          fgcolor: None,
12.          bgcolor: None,
13.      }
14.  }
15. }

```

7-15 fgcolor bgcolor Option Color
 fgcolor bgcolor Option T
 is_empty
 Option Color fgcolor bgcolor None
 Color From &str String Color
 7-16
 7-16 Color From


```

1. use std::convert::From;
2. use std::str::FromStr;
3. use std::string::String;
4. impl<'a> From<&'a str> for Color {
5.     fn from(src: &str) -> Self {
6.         src.parse().unwrap_or(Color::Red)
7.     }
8. }
9. impl From<String> for Color {
10.    fn from(src: String) -> Self {
11.        src.parse().unwrap_or(Color::Red)
12.    }
13. }
14. impl FromStr for Color {
15.    type Err = ();
16.    fn from_str(src: &str) -> Result<Self, Self::Err> {
17.        let src = src.to_lowercase();
18.        match src.as_ref() {
19.            "red" => Ok(Color::Red),
20.            "yellow" => Ok(Color::Yellow),
21.            "blue" => Ok(Color::Blue),
22.            _ => Err(()),
23.        }
24.    }
25. }

```

From std::convert::From std::str::FromStr std::string::String
 7-16 4 13
 Color &str String from parse
 FromStr 14 24 Color FromStr

FromStr from_str Result
 Self Self Err
 to_lowercase match
 Color match Err
 9

611 parse unwrap_or parse
Result Ok T unwrap Err T
Color Red

Colorize trait 7-17

7-17 Colorize

```
1. trait Colorize {  
2.     fn red(self) -> ColoredString;  
3.     fn yellow(self) -> ColoredString;  
4.     fn blue(self) -> ColoredString;  
5.     fn color<S: Into<Color>>(self, color: S) -> ColoredString;  
6.     fn on_red(self) -> ColoredString;  
7.     fn on_yellow(self) -> ColoredString;  
8.     fn on_blue(self) -> ColoredString;  
9.     fn on_color<S: Into<Color>>(self, color: S) -> ColoredString;  
10. }
```

Colorize 59 color
on_color

ColoredString & a str Colorize
7-18

7-18 ColoredString & a str Colorize

```

1. impl Colorize for ColoredString {
2.     fn red(self) -> ColoredString {self.color(Color::Red)}
3.     fn yellow(self) -> ColoredString {self.color(Color::Yellow)}
4.     fn blue(self) -> ColoredString {self.color(Color::Blue)}
5.     fn color<S: Into<Color>>(self, color: S) -> ColoredString {
6.         ColoredString { fgcolor: Some(color.into()), ..self }
7.     }
8.     fn on_red(self) -> ColoredString {self.on_color(Color::Red)}
9.     fn on_yellow(self) -> ColoredString {
10.        self.on_color(Color::Yellow)
11.    }
12.    fn on_blue(self) -> ColoredString {self.on_color(Color::Blue)}
13.    fn on_color<S: Into<Color>>(self, color: S) -> ColoredString {
14.        ColoredString { bgcolor: Some(color.into()), ..self }
15.    }
16. }
17. impl<'a> Colorize for &'a str {
18.     fn red(self) -> ColoredString {self.color(Color::Red)}
19.     fn yellow(self) -> ColoredString {self.color(Color::Yellow)}
20.     fn blue(self) -> ColoredString {self.color(Color::Blue)}
21.     fn color<S: Into<Color>>(self, color: S) -> ColoredString {
22.         ColoredString {
23.             fgcolor: Some(color.into()),
24.             input: String::from(self),
25.             ..ColoredString::default()
26.         }

```

```

27.     }
28.     fn on_red(self) -> ColoredString {self.on_color(Color::Red)}
29.     fn on_yellow(self) -> ColoredString {
30.         self.on_color(Color::Yellow)
31.     }
32.     fn on_blue(self) -> ColoredString {self.on_color(Color::Blue)}
33.     fn on_color<S: Into<Color>>(self, color: S) -> ColoredString {
34.         ColoredString {
35.             bgcolor: Some(color.into()),
36.             input: String::from(self),
37.             ..ColoredString::default()
38.         }
39.     }
40. }

```

7-18 `Color` `on_color` trait `Into<Color>` `Color::From` `String` `&a str` `into` `Color`

`compute_style` `ColoredString` `if` 7-19

7-19 `compute_style`

```

1. impl ColoredString {
2.     fn compute_style(&self) -> String {
3.         let mut res = String::from("\x1B[");
4.         let mut has_wrote = false;
5.         if let Some(ref bgcolor) = self.bgcolor {
6.             if has_wrote { res.push(';'); }
7.             res.push_str(bgcolor.to_bg_str());
8.             has_wrote = true;
9.         }
10.        if let Some(ref fgcolor) = self.fgcolor {
11.            if has_wrote { res.push(';'); }
12.            res.push_str(fgcolor.to_fg_str());
13.        }
14.        res.push('m');
15.        res
16.    }
17. }

```

compute_style 如果 let 有 main 7-20

7-20 main

```

1. fn main() {
2.     let red = "red".red();
3.     println!("{}", red);
4.     let yellow = "yellow".yellow().on_blue();
5.     println!("{}", yellow);
6.     let blue = "blue".blue();
7.     println!("{}", blue);
8.     let red = "red".color("red");
9.     println!("{}", red);
10.    let yellow = "yellow".on_color("yellow");
11.    println!("{}", yellow);
12. }

```

color on_color
color.rs

color.rs
trait
Rust

7.1.3

4
Rust

Newtype Drop 7-21

7-21 Drop

```
1. struct PrintDrop(&'static str);
2. impl Drop for PrintDrop {
3.     fn drop(&mut self) {
4.         println!("Dropping {}", self.0)
5.     }
6. }
```

7-21 Newtype
Newtype Rust
Newtype

- Newtype
- f64 Miles f64 Kilometers f64
- f64 Miles f64

7-21 Newtype

7-22

7-22

```
1. fn main() {
2.     let x = PrintDrop("x");
3.     let y = PrintDrop("y");
4. }
```

7-22 7-23

7-23 7-22

Dropping y
Dropping x

```

    x y
Rust

```

7-24

7-24

```
1. fn main() {
2.     let tup1 = (PrintDrop("a"), PrintDrop("b"), PrintDrop("c"));
3.     let tup2 = (PrintDrop("x"), PrintDrop("y"), PrintDrop("z"));
4. }
```

7-24 7-25

7-25 7-24

Dropping x
Dropping y
Dropping z
Dropping a
Dropping b
Dropping c

[illegible]

tup2 panic

7-26

7-26 tup2 panic

```
1. fn main() {
2.     let tup1 = (PrintDrop("a"), PrintDrop("b"), PrintDrop("c"));
3.     let tup2 = (PrintDrop("x"), PrintDrop("y"), panic!());
4. }
```

panic main

7-27 7-26

```
Dropping y
Dropping x
Dropping a
Dropping b
Dropping c
```

tup2 7-24

tup2 tup2

7-28

7-28

```
1. enum E{
2.     Foo(PrintDrop, PrintDrop)
3. }
4. struct Foo{
5.     x: PrintDrop,
6.     y: PrintDrop,
7.     z: PrintDrop,
8. }
9. fn main() {
10.    let e = E::Foo(PrintDrop("a"), PrintDrop("b"));
11.    let f = Foo{
12.        x: PrintDrop("x"), y: PrintDrop("y"), z: PrintDrop("z")
13.    };
14. }
```


7-28

7-29

Dropping x

Dropping y

Dropping z

Dropping a

Dropping b

f e
panic

Slice

7-30

7-30

```
1. fn main() {  
2.     let z = PrintDrop("z");  
3.     let x = PrintDrop("x");  
4.     let y = PrintDrop("y");  
5.     let closure = move || { y; z; x; };  
6. }
```

7-30

7-31

Dropping y

Dropping z

Dropping x

7-32

7-32

```

1. fn main() {
2.     let y = PrintDrop("y");
3.     let x = PrintDrop("x");
4.     let z = PrintDrop("z");
5.     let closure = move || {
6.         { let z_ref = &z; }
7.         x; y; z;
8.     };
9. }

```

7-32 z 7-33
7-33 **7-32**

Dropping z
Dropping x
Dropping y

7-30 z move
move z → x → y

7.2

trait Rust

GoF 23 20
20
4

-
-
-
-

rust trait 4 trait

Rust 6 Rust Rust

7.2.1

Rust Java 0 C++ Rust Java Rust Rust

Builder Pattern Rust 7-34

7-34

```
1. struct Circle {
2.     x: f64,
3.     y: f64,
4.     radius: f64,
5. }
6. struct CircleBuilder {
7.     x: f64,
8.     y: f64,
9.     radius: f64,
10. }
11. impl Circle {
12.     fn area(&self) -> f64 {
13.         std::f64::consts::PI * (self.radius * self.radius)
14.     }
15.     fn new() -> CircleBuilder {
16.         CircleBuilder {
17.             x: 0.0, y: 0.0, radius: 1.0,
18.         }
19.     }
20. }
21. impl CircleBuilder {
22.     fn x(&mut self, coordinate: f64) -> &mut CircleBuilder {
23.         self.x = coordinate;
24.         self
25.     }
26.     fn y(&mut self, coordinate: f64) -> &mut CircleBuilder {
27.         self.y = coordinate;
```

```

28.     self
29. }
30. fn radius(&mut self, radius: f64) -> &mut CircleBuilder {
31.     self.radius = radius;
32.     self
33. }
34. fn build(&self) -> Circle {
35.     Circle {
36.         x: self.x, y: self.y, radius: self.radius,
37.     }
38. }
39. }
40. fn main() {
41.     let c = Circle::new()
42.         .x(1.0).y(2.0).radius(2.0)
43.         .build();
44.     assert_eq!(c.area(), 12.566370614359172);
45.     assert_eq!(c.x, 1.0);
46.     assert_eq!(c.y, 2.0);
47. }

```

7-34 使用 builder 模式实现一个 Circle 结构体，并实现其 builder 模式。

1 使用 5 个方法实现 Circle 结构体，x、y、radius 三个属性。

6 使用 10 个方法实现 CircleBuilder 结构体，实现 Circle 结构体的 builder 模式。

11 使用 20 个方法实现 CircleBuilder 结构体，实现 Circle 结构体的 builder 模式，并实现 CircleBuilder 结构体的 build 方法。

21 使用 39 个方法实现 CircleBuilder 结构体，实现 Circle 结构体的 builder 模式，并实现 CircleBuilder 结构体的 build 方法。

main Circle::new(1.0, 2.0).radius(2.0).build Circle

Rust std::process::Command
7-35

7-35 std::process::Command

```
1. use std::process::Command;
2. fn main() {
3.     Command::new("ls")
4.         .arg("-l")
5.         .arg("-a")
6.         .spawn()
7.         .expect("ls command failed to start");
8. }
```

7-35 Commad 7-34 Circle

7.2.2

Rust Visitor Pattern Rust

Rust

Rust 7-36

7-36 Rust

```

1. mod ast {
2.     pub enum Stmt {
3.         Expr(Expr),
4.         Let(Name, Expr),
5.     }
6.     pub struct Name {
7.         value: String,
8.     }
9.     pub enum Expr {
10.        IntLit(i64),
11.        Add(Box<Expr>, Box<Expr>),
12.        Sub(Box<Expr>, Box<Expr>),
13.    }
14. }
15. mod visit {
16.     use ast::*;
17.     pub trait Visitor<T> {
18.         fn visit_name(&mut self, n: &Name) -> T;
19.         fn visit_stmt(&mut self, s: &Stmt) -> T;
20.         fn visit_expr(&mut self, e: &Expr) -> T;
21.     }
22. }

```

7-36 Rust
 `ast`
`Stmt`
`Name`
`Expr`
`mod`
 10

ast Rust

-
-

7-36 ast Visitor trait
Visitor trait
7-37

7-37 Visitor

```
1. use visit::*;
2. use ast::*;
3. struct Interpreter;
4. impl Visitor<i64> for Interpreter {
5.     fn visit_name(&mut self, n: &Name) -> i64 { panic!() }
6.     fn visit_stmt(&mut self, s: &Stmt) -> i64 {
7.         match *s {
8.             Stmt::Expr(ref e) => self.visit_expr(e),
9.             Stmt::Let(..) => unimplemented!(),
10.        }
11.    }
12.    fn visit_expr(&mut self, e: &Expr) -> i64 {
13.        match *e {
14.            Expr::IntLit(n) => n,
15.            Expr::Add(ref lhs, ref rhs) =>
16.                self.visit_expr(lhs) + self.visit_expr(rhs),
17.            Expr::Sub(ref lhs, ref rhs) =>
18.                self.visit_expr(lhs) - self.visit_expr(rhs),
19.        }
20.    }
21. }
```

7-37 Interpreter Visitor
Visitor

Serde

Serde Rust
Serialize Deserialize
Ser De Serde
JSON XML BinCode YAML MessagePack TOML

Serde 支持 Rust 的 `String` `option` `unit` `seq` `tuple` `tuple_struct` `map` `struct` `option` `Option` `T` `tuple_struct` `seq` `Vec` `T` `map` `k-v` `HashMap` `k` `v` Serde 支持

Serde trait 7-38

7-38 Serde trait

```

1. pub trait Deserialize<'de>: Sized {
2.     fn deserialize<D>(deserializer: D) -> Result<Self, D::Error>
3.     where D: Deserializer<'de>;
4. }
5. pub trait Deserializer<'de>: Sized {
6.     type Error: Error;
7.     fn deserialize_any<V>(self, visitor: V)
8.         -> Result<V::Value, Self::Error> where V: Visitor<'de>;
9.     fn deserialize_str<V>(self, visitor: V)
10.        -> Result<V::Value, Self::Error> where V: Visitor<'de>;
11.     ...
12. }
13. pub trait Visitor<'de>: Sized {
14.     type Value;
15.     fn visit_bool<E>(self, v: bool) -> Result<Self::Value, E>
16.         where E: Error,
17.     {
18.         Err(Error::invalid_type(Unexpected::Bool(v), &self))
19.     }
20.     fn visit_str<E>(self, v: &str) -> Result<Self::Value, E>
21.     where E: Error,
22.     {
23.         Err(Error::invalid_type(Unexpected::Str(v), &self))
24.     }
25.     ...
26. }
```

Figure 7-38: Implementing `Deserialize` and `Visitor` traits for `Serde`. The `Serde` trait defines methods for serializing and deserializing values. The `Deserialize` trait defines methods for deserializing values. The `Visitor` trait defines methods for visiting values.

Figure 7-39: Implementing `serde_json` for `Serde`. The `serde_json` module implements the `Serde` trait for JSON values. The `Value` enum represents JSON values.

Figure 7-39: `serde_json` for `Value`

```
1. #[derive(Debug, Clone, PartialEq)]
2. pub enum Value {
3.     Null,
4.     Bool(bool),
5.     Number(Number),
6.     String(String),
7.     Array(Vec<Value>),
8.     Object(Map<String, Value>),
9. }
```

Figure 7-39: Implementing `Value` for `serde_json`. The `Value` enum represents JSON values. The `serde_json` module implements the `Serde` trait for JSON values. The `Value` enum represents JSON values.

Figure 7-40: `serde_json` for `Visitor` and `Deserialize`

```
1. impl<'de> Deserialize<'de> for Value {
2.     fn deserialize<D>(deserializer: D) -> Result<Value, D::Error>
```

```

3.     where D: serde::Deserializer<'de>,
4.     {
5.         struct ValueVisitor;
6.         impl<'de> Visitor<'de> for ValueVisitor {
7.             type Value = Value;
8.             fn visit_bool<E>(self, value: bool) -> Result<Value, E> {
9.                 Ok(Value::Bool(value))
10.            }
11.            ...
12.        }
13.        deserializer.deserialize_any(ValueVisitor);
14.    }
15. }
16. impl<'de> serde::Deserializer<'de> for Value {
17.     type Error = Error;
18.     fn deserialize_any<V>(self, visitor: V)
19.         -> Result<V::Value, Error> where V: Visitor<'de>,
20.     {
21.         match self {
22.             Value::Null => visitor.visit_unit(),
23.             Value::Bool(v) => visitor.visit_bool(v),
24.             Value::Number(n) => n.deserialize_any(visitor),
25.             Value::String(v) => visitor.visit_string(v),
26.             Value::Array(v) => {visitor.visit_seq(...)},
27.             Value::Object(v) => { visitor.visit_map(...)}
28.         }
29.     }
30. }

```

7-40 `serde_json`
 Deserialize `deserialize`
 deserialize `ValueVisitor` `Visitor`
`serde_json` `Value` `serde` `Deserializer`
`deserialize_any` `Value` `match`
`Value` `visit_xxx`

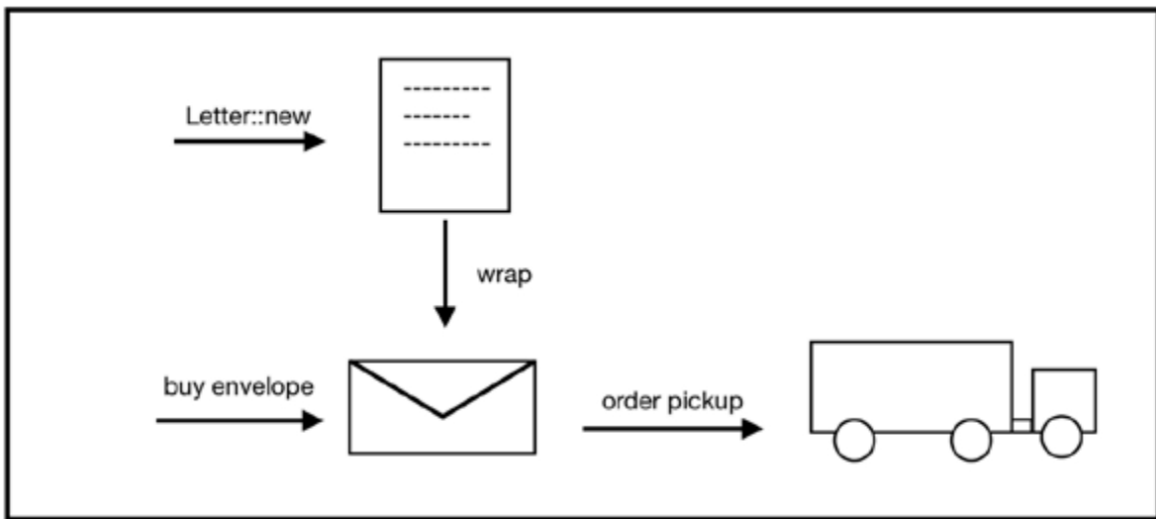

```
5. pub struct Envelope {
6.     letter: Option<Letter>,
7. }
8. pub struct PickupLorryHandle {
9.     done: bool,
10. }
11. impl Letter {
12.     pub fn new(text: String) -> Self {
13.         Letter {text: text}
14.     }
15. }
16. impl Envelope {
17.     pub fn wrap(&mut self, letter: &Letter){
18.         self.letter = Some(letter.clone());
19.     }
20. }
21. pub fn buy_prestamped_envelope() -> Envelope {
22.     Envelope {letter: None}
23. }
24. impl PickupLorryHandle {
25.     pub fn pickup(&mut self, envelope: &Envelope) {
26.         /*give letter*/
27.     }
28.     pub fn done(&mut self) {
29.         self.done = true;
30.         println!("sent");
31.     }
32. }
33. pub fn order_pickup() -> PickupLorryHandle {
34.     PickupLorryHandle {done: false , /* other handles */}
35. }
36. fn main(){
37.     let letter = Letter::new(String::from("Dear RustFest"));
38.     let mut envelope = buy_prestamped_envelope();
39.     envelope.wrap(&letter);
40.     let mut lorry = order_pickup();
41.     lorry.pickup(&envelope);
42.     lorry.done();
43. }
```

7-41 1 4 Letter
 5 7 Envelope Option Letter
 8 10 PickupLorryHandle
 bool done 11 20 Letter
 Envelope new wrap

21 23 buy_prestamped_envelope letter
 None Envelope

24 32 PickupLorryHandle pickup done
 33 35 order_pickup

7-3



7-3 7-41

main 7-42

7-42 main

```

1. fn main() {
2.     let letter = Letter::new(String::from("Dear RustFest"));
3.     let mut envelope = buy_prestamped_envelope();
4.     envelope.wrap(&letter);
5.     let mut lorry = order_pickup();
6.     lorry.pickup(&envelope);
7.     lorry.done();
8. }
    
```



```

17. pub fn buy_prestamped_envelope() -> EmptyEnvelope {
18.     EmptyEnvelope {}
19. }
20. impl PickupLorryHandle {
21.     pub fn pickup(&mut self, envelope: ClosedEnvelope) {
22.         /*give letter*/
23.     }
24.     pub fn done(self) {}
25. }
26. impl Drop for PickupLorryHandle {
27.     fn drop(&mut self) { println!("sent"); }
28. }
29. pub fn order_pickup() -> PickupLorryHandle {
30.     PickupLorryHandle {done: false , /* other handles */}
31. }
32. fn main(){
33.     let letter = Letter::new(String::from("Dear RustFest"));
34.     let envelope = buy_prestamped_envelope();
35.     let closed_envelope = envelope.wrap(letter);
36.     let mut lorry = order_pickup();
37.     lorry.pickup(closed_envelope);
38. }

```

7-43 Envelope EmptyEnvelope ClosedEnvelope wrap letter ClosedEnvelope letter buy_prestamped_envelope EmptyEnvelope

PickupLorryHandle pickup envelope ClosedEnvelope PickupLorryHandle Drop drop done

7-44 main

7-44 main


```

1. fn main(){
2.     let letter = Letter::new(String::from("Dear RustFest"));
3.     let envelope = buy_prestamped_envelope();
4.     let closed_envelope = envelope.wrap(letter);
5.     let mut lorry = order_pickup();
6.     lorry.pickup(closed_envelope);
7. }

```

[7-44](#) `sent` `PickupLorryHandle` `lorry`
`main` `drop` `RAII`

`RAII` `GoF` `Rust` `RAII`
`HTTP` `Rust`

7.3

`Rust` `Rust`
`trait` `Rust`
`Rust`

`Rust`
`RAII` `Rust`

`Rust` `Rust`
`RustConf 2018` [\[1\]](#)
`Data-Oriented` `Rust`
`ECS` `Generational Index`
`AnyMap` `Register`

[\[1\] RustConf 2018](#) <https://zhuanlan.zhihu.com/p/44657202>

第8章 数据类型

数据类型是编程语言中用于描述数据的类型。

数据类型可以分为基本数据类型和复合数据类型。基本数据类型包括 Pascal 中的 `Integer`、`Real`、`Boolean` 和 `Char`。复合数据类型包括 `Array`、`Record`、`Set` 和 `File`。Pascal 中的数据类型是静态的，即在编译时必须确定。

数据类型可以分为静态数据类型和动态数据类型。静态数据类型在编译时必须确定，而动态数据类型在运行时确定。Pascal 是静态类型语言，而 Python 是动态类型语言。

数据类型可以分为强类型和弱类型。强类型语言要求变量只能存储特定类型的数据，而弱类型语言允许变量存储任意类型的数据。Pascal 是强类型语言，而 Python 是弱类型语言。Rust 是一种强类型、静态类型语言，旨在提高程序的安全性和性能。

8.1 字符

字符是计算机中最小的数据单位。在计算机中，字符通常用 ASCII 码表示。ASCII 码是一个 7 位的二进制数，范围从 0 到 127。ASCII 码表中，大写字母 A 的 ASCII 码是 0100_0001，小写字母 a 的 ASCII 码是 0110_0001。Character Encoding 是指将字符转换为计算机可以存储和处理的二进制形式。


8.1.1 字符集

字符集是指一组字符的集合。ASCII 是最常见的字符集，它包含 128 个字符。Unicode 是一个更广泛的字符集，它包含来自世界各地的各种语言字符。UTF-8 是一种变长的字符编码方式，它可以将 Unicode 字符编码为 1 到 4 个字节。

在 Python 中，字符串是由 Unicode 字符组成的序列。Python 3 默认使用 UTF-8 编码。GB2312 是一种中文字符编码标准，它包含 6000 多个中文字符。

GB2312GBKGB2312
2GB18030
GB2312GBKGB2312
2GB18030
GB2312GBKGB2312
2GB18030
GB2312GBKGB2312
2GB18030

Unicode
Unicode
Code Point
Scala
Value
0x00000xD7FF 0xE0000x10FFFF
Unicode
Unicode
4Unicode
Unicode
Code Unit 8-1

A 英文字符	道 中文	 emoji
Code Point U+0x41	U+9053	U+1F600
UTF-8 Code Unit 0x41	0xE9 0x81 0x93	0xF0 0x9F 0x98 0x84
Byte 1	3	4

8-1

Unicode
Unicode
UTF-8UTF-16UTF-32
124
UTF-16UTF-32
24
UTF-81

UTF-8148-2

Unicode范围	UTF-8编码 (1~4字节)
U+ 0000 ~ U+ 007F	0XXXXXXX
U+ 0080 ~ U+ 07FF	110XXXXX 10XXXXXX
U+ 0800 ~ U+ FFFF	1110XXXX 10XXXXXX 10XXXXXX
U+10000 ~ U+1FFFF	11110XXX 10XXXXXX 10XXXXXX 10XXXXXX

8-2 UTF-8

UTF-8

- 的ASCII 的ASCII 1 的ASCII 7 0

- n n 1 $n + 1$ 0 10

8-1 “ ” U+9053 1001_0000_0101_0011 UTF-8 1110_1001_10_000001_10_010011 0xE90x810x93

Unicode Encode Decode

UTF-8 Unicode Unicode UTF-8

8-3

	道				
Unicode Code Point	U+	9	0	5	3
	1001 0000 0101 0011				
<div>编码 ↓</div> <div>↑ 解码</div>	1001 000001 010011 1110 XXXX 10 XXXXXX 10 XXXXXX				
UTF-8	1110 1001 10 000001 10 010011				
	0xE9 0x81 0x93				

图8-3 UTF-8的编码和解码

图8-3展示了Unicode到UTF-8的编码和解码过程。Unicode码点0x9053（十进制36867）被编码为UTF-8字节序列0xE9 0x81 0x93。该序列在内存中以二进制形式存储为0b111010011000000110010011。

图8-1展示了在Rust中处理UTF-8字符串的示例代码。

```
1. use std::str;
2. fn main() {
3.     let tao = str::from_utf8(&[0xE9u8, 0x81u8, 0x93u8]).unwrap();
4.     assert_eq!("道", tao);
5.     assert_eq!("道", String::from("\u{9053}"));
6.     let unicode_x = 0x9053;
7.     let utf_x_hex = 0xe98193;
8.     let utf_x_bin = 0b111010011000000110010011;
9.     println!("unicode_x: {:b}", unicode_x);
10.    println!("utf_x_hex: {:b}", utf_x_hex);
11.    println!("utf_x_bin: 0x{:x}", utf_x_bin);
12. }
```

图8-1展示了在Rust中处理UTF-8字符串的示例代码。代码中使用了`str::from_utf8`函数，该函数接受一个UTF-8字节序列（在这里是`&[0xE9u8, 0x81u8, 0x93u8]`）并返回一个`String`对象。代码还展示了如何将Unicode码点`0x9053`转换为UTF-8字节序列，并验证了转换结果。

0x0b UTF-8 println 8-3

8.1.2

Rust char char Unicode 8-2

8-2

```
1. fn main() {
2.     let tao = '道';
3.     let tao_u32 = tao as u32;
4.     assert_eq!(36947, tao_u32);
5.     println!("U+{:x}", tao_u32); // U+9053
6.     println!("{}", tao.escape_unicode()); // \u{9053}
7.     assert_eq!(char::from(65), 'A');
8.     assert_eq!(std::char::from_u32(0x9053), Some('道'));
9.     assert_eq!(std::char::from_u32(36947), Some('道'));
10.    assert_eq!(std::char::from_u32(12901010101), None);
11. }
```

8-2 Rust char u32 u32 Unicode 10 None

3 as char u32 tao u32 36947 5 println U+9053 “” Unicode char escape_unicode Unicode

Unicode Rust 4 8-3

8-3

```

1. fn main() {
2.     let mut b = [0; 3];
3.     let tao = '道';
4.     let tao_str = tao.encode_utf8(&mut b);
5.     assert_eq!("道", tao_str);
6.     assert_eq!(3, tao.len_utf8());
7. }

```

8-3 `b` `encode_utf8` `tao` `UTF-8` `b` `len_utf8` `tao` `UTF-8`

Unicode 8-4

8-4

```

1. fn main() {
2.     let e = 'é';
3.     println!("{}", e as u32);
4. }

```

8-4

error: character literal may only contain one codepoint: 'é'

```

2 |     let e = 'é';
  |               ^^^^

```

`e` `é` Rust 1.30

`char` 8-5

8-5

```

1. fn main(){
2.     assert_eq!(true, 'f'.is_digit(16));
3.     assert_eq!(Some(15), 'f'.to_digit(16));
4.     assert!('a'.is_lowercase());
5.     assert!(!'道'.is_lowercase());
6.     assert!(!'a'.is_uppercase());
7.     assert!('A'.is_uppercase());
8.     assert!(!'中'.is_uppercase());
9.     assert_eq!('i', 'I'.to_lowercase());
10.    assert_eq!('B', 'b'.to_uppercase());
11.    assert!(' '.is_whitespace());
12.    assert!('\u{A0}'.is_whitespace());
13.    assert!(!'越'.is_whitespace());
14.    assert!('a'.is_alphabetic());
15.    assert!('京'.is_alphabetic());
16.    assert!(!'1'.is_alphabetic());
17.    assert!('7'.is_alphanumeric());
18.    assert!('K'.is_alphanumeric());
19.    assert!('藏'.is_alphanumeric());
20.    assert!(!'¼'.is_alphanumeric());
21.    assert!('□'.is_control());
22.    assert!(!'q'.is_control());
23.    assert!('ㄗ'.is_numeric());
24.    assert!('7'.is_numeric());
25.    assert!(!'ᑭ'.is_numeric());
26.    assert!(!'藏'.is_numeric());
27.    println!("{}", '\r'.escape_default());
28. }

```

8-5

· is_digit16
 to_digit16

- `is_lowercase` 判断是否是 Unicode 小写字母
- `is_uppercase` 判断是否是 Unicode 大写字母
- `to_lowercase` Unicode 小写字母
- `to_uppercase` Unicode 大写字母
- `is_whitespace` 判断是否是空白符
- `is_alphabetic` 判断是否是字母
- `is_alphanumeric` 判断是否是字母或数字
- `is_control` 判断是否是控制符
- `is_numeric` 判断是否是数字
- `escape_default` 转义 \t \r \n 等字符

8.1.3 字符串

Unicode 字符串在 Rust 中以 UTF-8 编码，Rust 字符串是 UTF-8 编码的 Unicode 字符串。

- **str** 字符串切片
- **String** 字符串
- **CStr** C 字符串 Rust 字符串 C 字符串
- **CString** Rust 字符串 C 字符串 C 字符串 Rust 字符串
- **OsStr** 操作系统字符串 Windows 字符串
- **OsString** OsStr 字符串 Rust 字符串
- **Path** 标准库 path 字符串 Path 字符串 OsStr
- **PathBuf** Path 字符串 Path 字符串 PathBuf 字符串 OsString

Rust 字符串切片 `str` `String` 3 个字符串切片 `str` 字符串切片 `DST` 字符串切片 `str` `Slice` 字符串切片 `&str`

&str UTF-8 &str

· static str

· &str String String &str

· str from_utf8 [u8 N] &str 8-1

&str String &str String String String Vec u8 String as_ptr len capacity 8-6

8-6 String

```
1. fn main() {
2.     let mut a = String::from("fooα");
3.     println!("{:p}", a.as_ptr());
4.     println!("{:p}", &a);
5.     assert_eq!(a.len(), 5);
6.     a.reserve(10);
7.     assert_eq!(a.capacity(), 15);
8. }
```

8-6 as_ptr &a

5 len

6 reserve 10 7 capacity 15 5

Rust &str String 8-7

8-7

```

1. fn main() {
2.     let string: String = String::new();
3.     assert_eq!("", string);
4.     let string: String = String::from("hello rust");
5.     assert_eq!("hello rust", string);
6.     let string: String = String::with_capacity(20);
7.     assert_eq!("", string);
8.     let str: &'static str = "the tao of rust";
9.     let string: String =
10.         str.chars().filter(|c| !c.is_whitespace()).collect();
11.     assert_eq!("thetaoofrust", string);
12.     let string: String = str.to_owned();
13.     assert_eq!("the tao of rust", string);
14.     let string: String = str.to_string();
15.     let str: &str = &string[11..15];
16.     assert_eq!("rust", str);
17. }

```

8-7 2 String new

4 String from String From trait

6 String with_capacity String new with_capacity usize 20 20

8 &static str 9 8 str chars collect String chars FromIterator trait

12 14 to_owned to_string &str String to_owned &str String to_string String from

15 String 11 14

8.1.4

Rust UTF-8
Rust bytes chars
Rust

chars bytes 8-8

8-8 chars bytes

```
1. fn main() {  
2.     let str = "borös";  
3.     let mut chars = str.chars();  
4.     assert_eq!(Some('b'), chars.next());  
5.     assert_eq!(Some('o'), chars.next());  
6.     assert_eq!(Some('r'), chars.next());  
7.     assert_eq!(Some('ö'), chars.next());  
8.     assert_eq!(Some('s'), chars.next());  
9.     let mut bytes = str.bytes();  
10.    assert_eq!(6, str.len());  
11.    assert_eq!(Some(98), bytes.next());  
12.    assert_eq!(Some(111), bytes.next());  
13.    assert_eq!(Some(114), bytes.next());  
14.    assert_eq!(Some(195), bytes.next());  
15.    assert_eq!(Some(182), bytes.next());  
16.    assert_eq!(Some(115), bytes.next());  
17. }
```

8-8 3 chars Chars Chars next
9 bytes Bytes Bytes next
10 len

Rust get get_mut
Rust UTF-8
8-9

8-9 `get` `get_mut`

```
1. fn main() {
2.     let mut v = String::from("borös");
3.     assert_eq!(Some("b"), v.get(0..1));
4.     assert_eq!(Some("ö"), v.get(3..5));
5.     assert_eq!(Some("orös"), v.get(1..));
6.     assert!(v.get_mut(4..).is_none());
7.     assert!(!v.is_char_boundary(4));
8.     assert!(v.get_mut(..8).is_none());
9.     assert!(v.get_mut(..42).is_none());
10. }
```

8-9 `String` `String` 3
6 `get` `Option` 6
4 4 `ö` `ö`
UTF-8 Rust `None`
`is_char_boundary` 7 4

`split_at` `split_at_mut`
8-10

8-10 `split_at`

```
1. fn main() {
2.     let s = "Per Martin-Löf";
3.     let (first, last) = s.split_at(12);
4.     assert_eq!("Per Martin-L", first);
5.     assert_eq!("öf", last);
6.     // 'main' panicked: byte index 13 is not a char boundary
7.     // let (first, last) = s.split_at(13);
8. }
```

8-10 `split_at` 3 12
7 13
`ö`

8.1.5 字符串

字符串在 Rust 中用 `String` 类型表示，它是一个不可变字符串。在 5.1 节中，我们学习了 `String` 类型。

字符串

在 Rust 中，`push` 和 `push_str` 方法用于向字符串添加内容。8-11 节

展示了 `8-11` 节中的 `push` 和 `push_str` 方法。

```
1. fn main() {
2.     let mut hello = String::from("Hello, ");
3.     hello.push('R');
4.     hello.push_str("ust!");
5.     assert_eq!("Hello, Rust!", hello);
6. }
```

在 8-11 节中，我们学习了 `push` 和 `String` 类型。在 `hello` 变量上调用 `push_str` 方法，将 `&str` 类型的字符串添加到 `hello` 字符串的末尾。在 `String` 类型中，`Vec` 类型的 `u8` 字节切片用于存储字符串。在 `push` 方法中，我们使用 `u8` 类型的 `Vec` 切片来存储 UTF-8 编码的字符串。在 `extend_from_slice` 方法中，我们使用 `push_str` 方法将 `&str` 类型的字符串添加到 `Vec` 切片中。在 `extend_from_slice` 方法中，我们使用 `String` 类型的 `Vec` 切片。

在 Rust 中，字符串是不可变的。在 `String` 类型中，我们使用 `Extend` trait 来扩展字符串。8-12 节

展示了 `8-12` 节中的 `Extend` trait。

```
1. fn main() {
2.     let mut message = String::from("hello");
3.     message.extend([' ', 'r', 'u'].iter());
4.     message.extend("st ".chars());
5.     message.extend("w o r l d".split_whitespace());
6.     assert_eq!("hello, rust world", &message);
7. }
```

在 8-12 节中，我们学习了 `String` 类型的 `Extend` trait。在 `extend` 方法中，我们使用 `iter` 方法将 `3` 个字符添加到字符串中。在 `extend` 方法中，我们使用 `4` 个字符添加到字符串中。

chars 和 Chars 都是 5 个 split_whitespace 和 SplitWhitespace

insert 和 insert_str 和 push 和 push_str 8-13

8-13 insert 和 insert_str

```
1. fn main() {
2.     let mut s = String::with_capacity(3);
3.     s.insert(0, 'f');
4.     s.insert(1, 'o');
5.     s.insert(2, 'o');
6.     s.insert_str(0, "bar");
7.     assert_eq!("barfoo", s);
8. }
```

8-13 insert 和 insert_str 和 is_char_boundary

String 的 Add 和 AddAssign 和 trait 8-14

8-14 "+" 和 "+="

```
1. fn main() {
2.     let left = "the tao".to_string();
3.     let mut right = "Rust".to_string();
4.     assert_eq!(left + " of " + &right, "the tao of Rust");
5.     right += "!";
6.     assert_eq!(right, "Rust!");
7. }
```

8-14 "+" 和 "+=" 和 &str 和 String 和 Deref trait

8-16

8-16

```
1. fn main() {
2.     let s = String::from("fooαbar");
3.     let s: String = s.chars().enumerate().map(|(i, c)| {
4.         if i % 2 == 0 {
5.             c.to_lowercase().to_string()
6.         } else {
7.             c.to_uppercase().to_string()
8.         }
9.     }).collect();
10.    assert_eq!("fOoAbAr", s);
11. }
```

8-16 chars Chars enumerate map collect String

Rust std::string 8-17

8-17

```

1. fn main() {
2.     let mut s = String::from("hαllo");
3.     s.remove(3);
4.     assert_eq!("hαlo", s);
5.     assert_eq!(Some('o'), s.pop());
6.     assert_eq!(Some('l'), s.pop());
7.     assert_eq!(Some('α'), s.pop());
8.     assert_eq!("h", s);
9.     let mut s = String::from("hαllo");
10.    s.truncate(3);
11.    assert_eq!("hα", s);
12.    s.clear();
13.    assert_eq!(s, "");
14.    let mut s = String::from("α is alpha, β is beta");
15.    let beta_offset = s.find('β').unwrap_or(s.len());
16.    let t: String = s.drain(..beta_offset).collect();
17.    assert_eq!(t, "α is alpha, ");
18.    assert_eq!(s, "β is beta");
19.    s.drain(..);
20.    assert_eq!(s, "");
21. }

```

8-17

remove 3
remove
32

57 pop 8

10 truncate
truncate 33αα
s "hα" truncate
clear truncate

12 clear truncate truncate
0 clear

14 drain 15 find
β 16
drain Drain Drain

8.1.6

Rust DSL
Rust C
Rust Rust **regex**
Rust
20 20

- contains starts_with ends_with
 - find rfind
 - split rsplit split_terminator rsplit_terminator splitn rsplitn
 - matches rmatches match_indices rmatch_indices
 - trim_matches trim_left_matches trim_right_matches
 - replace replacen
- bool

8-18

8-18 contains

```

1. fn main() {
2.     let bananas = "bananas";
3.     assert!(bananas.contains('a'));
4.     assert!(bananas.contains("an"));
5.     assert!(bananas.contains(char::is_lowercase));
6.     assert!(bananas.starts_with('b'));
7.     assert!(!bananas.ends_with("nana"));
8. }

```

8-18 3 5 contains char &str fn pointer contains 8-19 std str contains

8-19 std str contains

```

1. pub fn contains<'a, P: Pattern<'a>>(&'a self, pat: P) -> bool {
2.     core_str::StrExt::contains(self, pat)
3. }

```

8-19 contains pat Pattern a
 Pattern a & a str trait Rust char
 String &str &&str &[char] FnMut char - bool
 trait contains

8-18 6 7 starts_with ends_with
 contains Pattern a
 pattern starts_with ends_with pattern

find 8-20

8-20 find

```

1. fn main() {
2.     let s = "Löwe 老虎 Léopard";
3.     assert_eq!(s.find('w'), Some(3));
4.     assert_eq!(s.find('老'), Some(6));
5.     assert_eq!(s.find('虎'), Some(9));
6.     assert_eq!(s.find("é"), Some(14));
7.     assert_eq!(s.find("Léopard"), Some(13));
8.     assert_eq!(s.rfind('L'), Some(13));
9.     assert_eq!(s.find(char::is_whitespace), Some(5));
10.    assert_eq!(s.find(char::is_lowercase), Some(1));
11. }

```

```

    find(pattern, 8-20).find(
        Option::usize,
        None, 8, rfind, r, right,
        Some(13)

```

```

    0000

```

```

    00000000000000000000 split 0000000000 8-21 000

```

```

    00008-21split00000000

```

```

1. fn main() {
2.     let s = "Löwe 虎 Léopard";
3.     let v = s.split(|c|
4.         (c as u32) >= (0x4E00 as u32) && (c as u32) <= (0x9FA5 as u32)
5.     ).collect::<Vec<&str>>();
6.     assert_eq!(v, ["Löwe ", " Léopard"]);
7.     let v = "abcdefXghi".split(|c|
8.         c == 'l' || c == 'X'
9.     ).collect::<Vec<&str>>();
10.    assert_eq!(v, ["abc", "def", "ghi"]);
11.    let v = "Mary had a little lambda"
12.        .splitn(3, ' ')
13.        .collect::<Vec<&str>>();
14.    assert_eq!(v, ["Mary", "had", "a little lambda"]);
15.    let v = "A.B.".split(".").collect::<Vec<&str>>();
16.    assert_eq!(v, ["A", "B", ""]);
17.    let v = "A.B.".split_terminator('.').collect::<Vec<&str>>();
18.    assert_eq!(v, ["A", "B"]);
19.    let v = "A..B..".split(".").collect::<Vec<&str>>();
20.    assert_eq!(v, ["A", "", "B", "", ""]);
21.    let v = "A..B..".split_terminator(".").collect::<Vec<&str>>();
22.    assert_eq!(v, ["A", "", "B", ""]);
23. }

```

8-212&str

35splitssplitpattern
 6 Vec&strU+4E00U+9FA5
 Rust u32as u32

79 10

1113splitnsplitnnpattern

14 3 3

15 22 split split_terminator terminator split_terminator

rsplit rsplitn rsplit_terminator lsplit split collect

Rust pattern matches 8-22

8-22 matches

```
1. fn main() {
2.     let v = "abcXXXabcYYYabc"
3.     .matches("abc").collect::<Vec<&str>>();
4.     assert_eq!(v, ["abc", "abc", "abc"]);
5.     let v = "1abc2abc3"
6.     .rmatches(char::is_numeric).collect::<Vec<&str>>();
7.     assert_eq!(v, ["3", "2", "1"]);
8.     let v = "abcXXXabcYYYabc"
9.     .match_indices("abc").collect::<Vec<_>>();
10.    assert_eq!(v, [(0, "abc"), (6, "abc"), (12, "abc")]);
11.    let v = "abcXXXabcYYYabc"
12.    .rmatch_indices("abc").collect::<Vec<_>>();
13.    assert_eq!(v, [(12, "abc"), (6, "abc"), (0, "abc")]);
14. }
```

8-22 2 3 matches collect Vec &str 4

5 6 rmatches

8 9 match_indices indices index

通过调用 `trim_matches` 函数，我们可以得到匹配结果，并返回一个 `Vec` 类型的 `_indices` 数组，该数组包含所有匹配的索引。

```
11 let mut indices = Vec::new();
12 for (i, _) in matches.iter().enumerate() {
13     indices.push(i);
14 }
```

通过调用 `std::str::trim` 函数，我们可以得到一个 `String` 类型的字符串，该字符串是原始字符串去除首尾空格后的结果。

8-23 trim

```
1. fn main() {
2.     let s = " Hello\tworld\t";
3.     assert_eq!("Hello\tworld", s.trim());
4.     assert_eq!("Hello\tworld\t", s.trim_left());
5.     assert_eq!(" Hello\tworld", s.trim_right());
6. }
```

通过调用 `trim` 函数，我们可以得到一个 `String` 类型的字符串，该字符串是原始字符串去除首尾空格后的结果。通过调用 `trim_left` 函数，我们可以得到一个 `String` 类型的字符串，该字符串是原始字符串去除左侧空格后的结果。通过调用 `trim_right` 函数，我们可以得到一个 `String` 类型的字符串，该字符串是原始字符串去除右侧空格后的结果。

通过调用 `trim_left` 函数，我们可以得到一个 `String` 类型的字符串，该字符串是原始字符串去除左侧空格后的结果。通过调用 `trim_right` 函数，我们可以得到一个 `String` 类型的字符串，该字符串是原始字符串去除右侧空格后的结果。通过调用 `trim_matches` 函数，我们可以得到一个 `Vec` 类型的 `_indices` 数组，该数组包含所有匹配的索引。

8-24 trim_matches


```

1. fn main() {
2.     assert_eq!("Hello\tworld\t".trim_matches('\t'), "Helloworld");
3.     assert_eq!("11foobar11".trim_matches('1'), "foobar");
4.     assert_eq!("123foobar123"
5.         .trim_matches(char::is_numeric), "foobar");
6.     let x: &[char] = &['1', '2'];
7.     assert_eq!("12foobar12".trim_matches(x), "foobar");
8.     assert_eq!(
9.         "1foobarXX".trim_matches(|c| c == '1' || c == 'X'),
10.        "foobar"
11.    );
12.    assert_eq!("11foobar11".trim_left_matches('1'), "foobar11");
13.    assert_eq!(
14.        "123foobar123".trim_left_matches(char::is_numeric),
15.        "foobar123");
16.    let x: &[char] = &['1', '2'];
17.    assert_eq!("12foobar12".trim_left_matches(x), "foobar12");
18.    assert_eq!(
19.        "1fooX".trim_right_matches(|c| c == '1' || c == 'X'),
20.        "1foo"
21.    );
22. }

```

8-24 trim_matches trim pattern pattern

211 trim_matches pattern

1221 trim_left_matches trim_right_matches

trim_matches replace 8-25

8-25 replace

```

1. fn main() {
2.     let s = "Hello\tworld\t";
3.     assert_eq!("Hello world ", s.replace("\t", " "));
4.     assert_eq!("Hello world", s.replace("\t", " ").trim());
5.     let s = "this is old old 123";
6.     assert_eq!("this is new new 123", s.replace("old", "new"));
7.     assert_eq!("this is new old 123", s.replacen("old", "new", 1));
8.     assert_eq!("this is ald ald 123", s.replacen('o', "a", 3));
9.     assert_eq!(
10.         "this is old old new23",
11.         s.replacen(char::is_numeric, "new", 1)
12.     );
13. }

```

8-25 3

trim 4

replace pattern

replacen 7 12

Rust

matches 8-26 matches

8-26 matches

```

1. fn matches<'a, P: Pattern<'a>>(&'a self, pat: P) -> Matches<'a, P>
2. {
3.     Matches(MatchesInternal(pat.into_searcher(self)))
4. }

```

8-26 matches Matches a P

8-27

8-27 Matches

```

1. struct MatchesInternal<'a, P: Pattern<'a>>(P::Searcher);
2. pub struct Matches<'a, P: Pattern<'a>>(MatchesInternal<'a, P>);
3. impl<'a, P: Pattern<'a>> Iterator for Matches<'a, P> {
4.     type Item = &'a str;
5.     fn next(&mut self) -> Option<&'a str> {
6.         self.0.next()
7.     }
8. }

```

□ □ □ □ □ 8-27 □ □ □ □ □ □ □ □ □ □ □ □ □ □ 2 □ 7 □ □ □ □ □ □ □ □
generate_pattern_iterators□ □ □ □ □ □ □ □

Matches □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ NewType □ □ □ □ □ □ □ □
MatchesInternal□ □ □ □ □ □ □ □3□ 7 □ □ □Matches□ □ □Iterator□ □ □ □ □ □ □ □
□ □ □next□ □ □ □ □ □ □ □ □ □MatchesInternal□ □ □ □next□ □ □ □ □ □ □ □6□ □ □ □ □
□ □ □ □8-28□ □ □MatchesInternal□ □next□next_back□ □ □ □ □ □ □
□ □ □ □8-28□**MatchesInternal**□ □**next**□**next_back**□ □ □ □ □ □ □

```

1. impl<'a, P: Pattern<'a>> MatchesInternal<'a, P> {
2.     fn next(&mut self) -> Option<&'a str> {
3.         self.0.next_match().map(|(a, b)| unsafe {
4.             self.0.haystack().slice_unchecked(a, b)
5.         })
6.     }
7.     fn next_back(&mut self) -> Option<&'a str>
8.     where P::Searcher: ReverseSearcher<'a>
9.     {
10.         self.0.next_match_back().map(|(a, b)| unsafe {
11.             self.0.haystack().slice_unchecked(a, b)
12.         })
13.     }
14. }

```

MatchesInternal □ □ □ □ □ NewType □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ P□ □ □
Searcher□ □ □ next □next_back□ □ □ □ □ □ □ □ □ □ □ □ □ □ □P□ □ □Searcher□ □ □
next_match□next_match_back□ □ □ □ □ □ □ □ □ □ □ □ □ □ □Map□ □ □ □ □ □ □ □ □ □ □ □ □ □ □collect□ □ □ □ □
Matches□ □ □ □ □ □ □ □ □ □ □ □ □ □ □8-4□ □ □ □ □ □ □ □ □ □ □ □ □ □ □

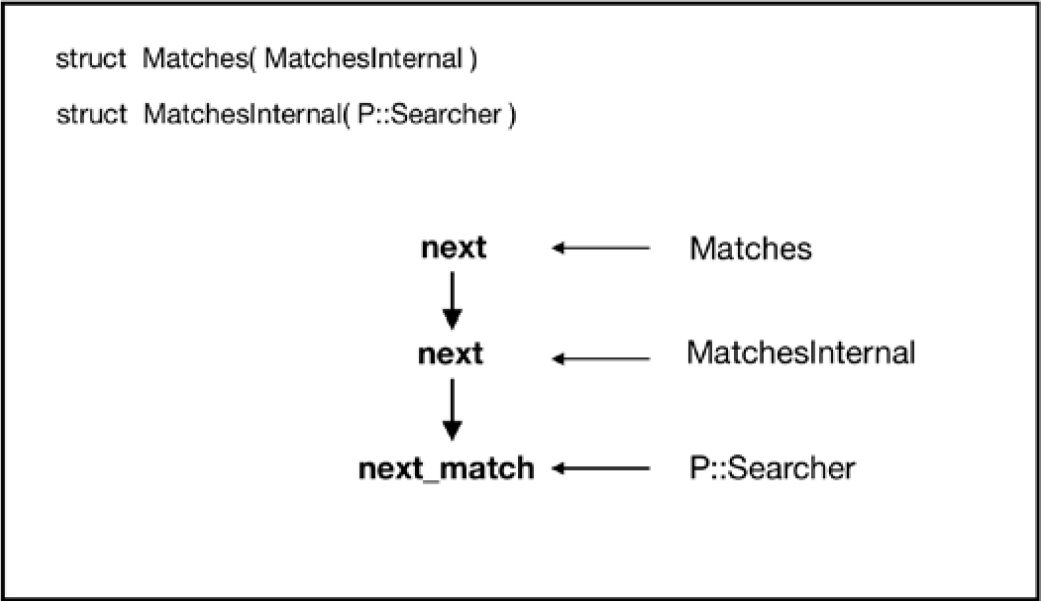


图8-4 Matches的内部结构

在Pattern trait中定义SearchStep枚举

图8-29 Pattern trait中的SearchStep枚举

```
1. pub enum SearchStep {
2.     Match(usize, usize),
3.     Reject(usize, usize),
4.     Done
```

```

5.  }
6.  pub trait Pattern<'a>: Sized {
7.      type Searcher: Searcher<'a>;
8.      fn into_searcher(self, haystack: &'a str) -> Self::Searcher;
9.      fn is_contained_in(self, haystack: &'a str) -> bool {
10.         self.into_searcher(haystack).next_match().is_some()
11.     }
12.     fn is_prefix_of(self, haystack: &'a str) -> bool {
13.         match self.into_searcher(haystack).next() {
14.             SearchStep::Match(0, _) => true,
15.             _ => false,
16.         }
17.     }
18.     fn is_suffix_of(self, haystack: &'a str) -> bool
19.     where Self::Searcher: ReverseSearcher<'a>
20.     {
21.         match self.into_searcher(haystack).next_back() {
22.             SearchStep::Match(_, j) if haystack.len() == j => true,
23.             _ => false,
24.         }
25.     }
26. }

```

8-29 Pattern a into_searcher haystack
 Searcher a trait into_searcher haystack
 “find a needle in a haystack” “”
 haystack needle “nana”
 “banana” “banana” haystack “nana” needle
 “haystack” “needle”

SearchStep Match usize usize
 haystack[0..3] Reject usize Done
 Done

is_contained_in needle haystack
is_prefix_of is_suffix_of KMP
14 22

KMP Rust
KMP **Two-Way**
KMP $O(n)$

8-30 Searcher a

8-30 Searcher a

```
1. pub unsafe trait Searcher<'a> {
2.     fn haystack(&self) -> &'a str;
3.     fn next(&mut self) -> SearchStep;
4.     fn next_match(&mut self) -> Option<(usize, usize)> {
5.         loop {
6.             match self.next() {
7.                 SearchStep::Match(a, b) => return Some((a, b)),
8.                 SearchStep::Done => return None,
9.                 _ => continue,
10.            }
11.        }
12.    }
13.    fn next_reject(&mut self) -> Option<(usize, usize)> {
14.        loop {
15.            match self.next() {
16.                SearchStep::Reject(a, b) => return Some((a, b)),
17.                SearchStep::Done => return None,
18.                _ => continue,
19.            }
20.        }
21.    }
22. }
```

8-30 Searcher a 2 haystack haystack

3 next SearchStep needle "aaaa"
haystack "cbaaaaab" next "[Reject 0 1 Reject
1 2 Match 2 5 Reject 5 8]"

4 21 next_match next_reject
SearchStep Option usize
usize

8-31 &a str Pattern a

8-31 &a str Pattern a

```
1. impl<'a, 'b> Pattern<'a> for &'b str {
2.     type Searcher = StrSearcher<'a, 'b>;
3.     fn into_searcher(self, haystack: &'a str) -> StrSearcher<'a, 'b>
4.     {
5.         StrSearcher::new(haystack, self)
6.     }
7.     fn is_prefix_of(self, haystack: &'a str) -> bool {
8.         haystack.is_char_boundary(self.len()) &&
9.         self == &haystack[..self.len()]
10.    }
11.    fn is_suffix_of(self, haystack: &'a str) -> bool {
12.        self.len() <= haystack.len() &&
13.        haystack.is_char_boundary(haystack.len() - self.len()) &&
14.        self == &haystack[haystack.len() - self.len()..]
15.    }
16. }
17. pub struct StrSearcher<'a, 'b> {
18.     haystack: &'a str,
19.     needle: &'b str,
20.     searcher: StrSearcherImpl,
```

```

21. }
22. enum StrSearcherImpl {
23.     Empty(EmptyNeedle),
24.     TwoWay(TwoWaySearcher),
25. }
26. unsafe impl<'a, 'b> Searcher<'a> for StrSearcher<'a, 'b> {
27.     fn haystack(&self) -> &'a str {
28.         self.haystack
29.     }
30.     fn next(&mut self) -> SearchStep {
31.         match self.searcher {
32.             StrSearcherImpl::Empty(ref mut searcher) => {...}
33.             StrSearcherImpl::TwoWay(ref mut searcher) => {...}
34.         }
35.     }
36.     fn next_match(&mut self) -> Option<(usize, usize)> {
37.         match self.searcher {
38.             StrSearcherImpl::Empty(..) => {...}
39.             StrSearcherImpl::TwoWay(ref mut searcher) => {...}
40.         }
41.     }
42. }

```

8-31 into_searcher & 'a str
 StrSearcher 'a 'b
 haystack needle haystack needle searcher
 StrSearcherImpl

StrSearcherImpl Empty EmptyNeedle
 TwoWay TwoWaySearcher
 EmptyNeedle
 TwoWaySearcher TwoWaySearch

Pattern 'a Searcher 'a
 SearchStep Rust

8.1.7 字符串解析

在 Rust 中，字符串解析通常使用 `std::str::parse` 函数。以下是一个简单的例子：

字符串解析

`std::str::parse` 函数用于解析字符串。以下是一个简单的例子：

8-32 `parse` 函数

```
1. fn main() {
2.     let four: u32 = "4".parse().unwrap();
3.     assert_eq!(4, four);
4.     let four = "4".parse::<u32>();
5.     assert_eq!(Ok(4), four);
6. }
```

在 `8-32` 中，`parse` 函数用于解析字符串 `"4"` 为 `u32` 类型。以下是一个简单的例子：

`parse` 函数返回 `FromStr` trait 的实现。以下是一个简单的例子：

8-33 `FromStr` trait

```
1. pub trait FromStr {
2.     type Err;
3.     fn from_str(s: &str) -> Result<Self, Self::Err>;
4. }
```

在 `8-33` 中，`FromStr` trait 用于定义字符串解析的接口。以下是一个简单的例子：

8-34 `FromStr` trait 的实现

```

1. use std::str::FromStr;
2. use std::num::ParseIntError;
3. #[derive(Debug, PartialEq)]
4. struct Point {
5.     x: i32,
6.     y: i32
7. }
8. impl FromStr for Point {
9.     type Err = ParseIntError;
10.    fn from_str(s: &str) -> Result<Self, Self::Err> {
11.        let coords = s.trim_matches(|p| p == '{' || p == '}' )
12.            .split(",")
13.            .collect::<Vec<&str>>();
14.        let x_fromstr = coords[0].parse::<i32>()?;
15.        let y_fromstr = coords[1].parse::<i32>()?;
16.        Ok(Point { x: x_fromstr, y: y_fromstr })
17.    }
18. }
19. fn main(){
20.    let p = Point::from_str("{1,2}");
21.    assert_eq!(p.unwrap(), Point{ x: 1, y: 2} );
22.    let p = Point::from_str("{3,u}");
23.    // Err(ParseIntError { kind: InvalidDigit })
24.    println!("{:?}", p);
25. }

```

8-34 Point
 11 16 trim_matches split
 Vec &str Point
 Result

22 Err
 ParseIntError{kind InvalidDigit}
 8

8-5 format

format 関数の基本的な使い方

- ・ `format!("{}", number)` で `number` を 8-35 の 4 番目の `number` の位置に置き換える。 `number` の位置は 6 番目

- ・ `format!("{:.number}", number)` で `number` の位置に `."` の `number` の位置に置き換える。 8-35 の 5 番目

- ・ `format!("{:width}{}", value, format!("{}", value))` で 8-35 の 7 番目の `10` の位置に置き換える。

8-35 の 11 番目の `12` の位置に `"="` の `"*` の位置に `format!("{}", value)` の位置に `13` の位置に `Unicode` の位置に置き換える。

8-36 の `Rust` の位置に `format!("{}", value)` の位置に置き換える。

8-36 format

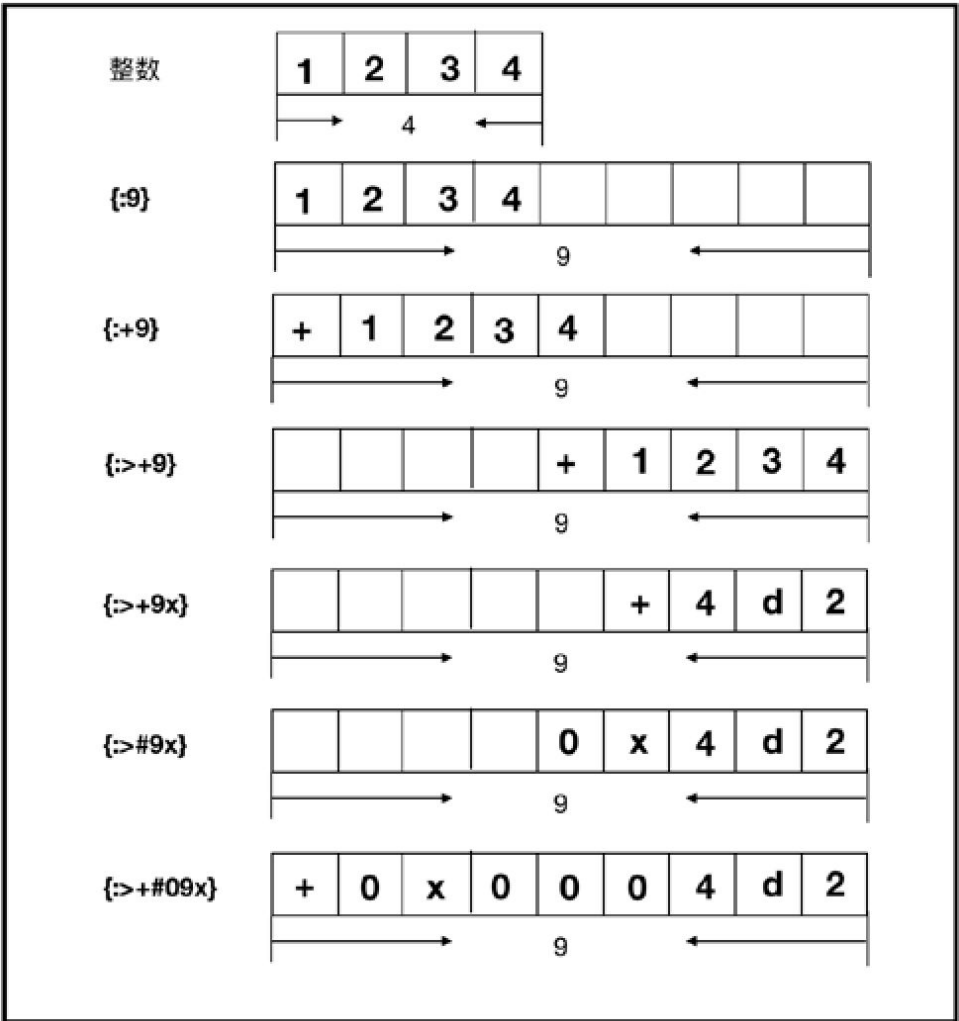
```
1. fn main() {
2.     assert_eq!(format!("{}", 1234), "+1234");
3.     assert_eq!(format!("{:x}", 1234), "+4d2");
4.     assert_eq!(format!("{:#x}", 1234), "+0x4d2");
5.     assert_eq!(format!("{:b}", 1234), "10011010010");
6.     assert_eq!(format!("{:#b}", 1234), "0b10011010010");
7.     assert_eq!(format!("{:#20b}", 1234), "0b10011010010");
8.     assert_eq!(format!("{:<#20b}", 1234), "0b10011010010");
9.     assert_eq!(format!("{:^#20b}", 1234), "0b10011010010");
10.    assert_eq!(format!("{:>#15x}", 1234), "+0x4d2");
11.    assert_eq!(format!("{:>#015x}", 1234), "+0x0000000004d2");
12. }
```

8-36 の `format` の位置に置き換える。

- ・ `+` の位置に置き換える。
- ・ `0x` の位置に置き換える。

• **0000000000000000**

8-6 format



□8-6□□□□□format□□□□□

☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐8-37☐☐

8-37 format

```

1. fn main(){
2.     assert_eq!(format!("{:.4}", 1234.5678), "1234.5678");
3.     assert_eq!(format!("{:.2}", 1234.5618), "1234.56");
4.     assert_eq!(format!("{:.2}", 1234.5678), "1234.57");
5.     assert_eq!(format!("{:<10.4}", 1234.5678), "1234.5678 ");
6.     assert_eq!(format!("{:^12.2}", 1234.5618), " 1234.56  ");
7.     assert_eq!(format!("{:0^12.2}", 1234.5678), "001234.57000");
8.     assert_eq!(format!("{:e}", 1234.5678), "1.2345678e3");
9. }

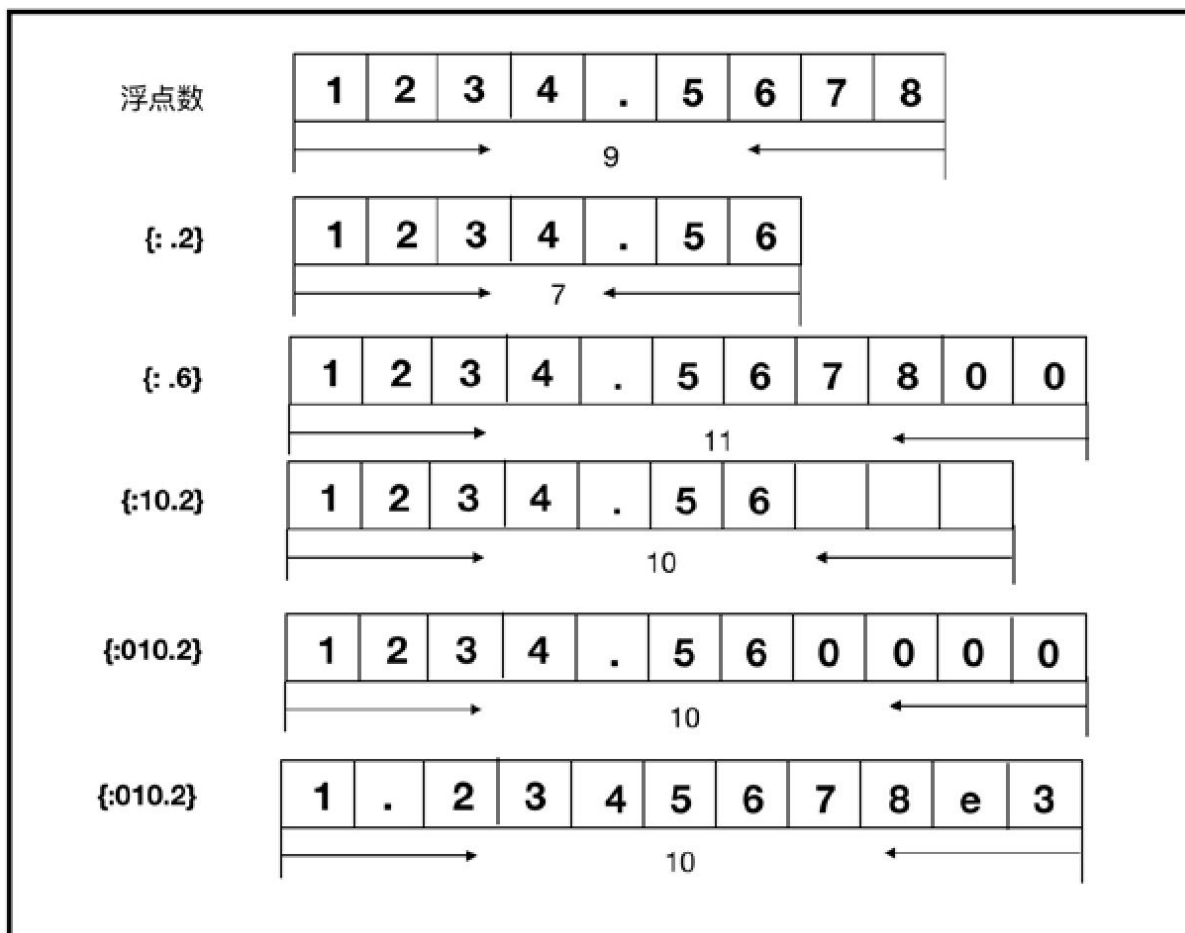
```

浮点数格式化的规则

· 浮点数格式化的规则是“.”后面的数字位数，如果位数不足，则在后面补0，如果位数超过，则四舍五入。

· 浮点数格式化的格式是 **{e}**，其中e表示科学计数法。

8-7 浮点数的格式化输出



8-7 format

println write Display trait 8-38

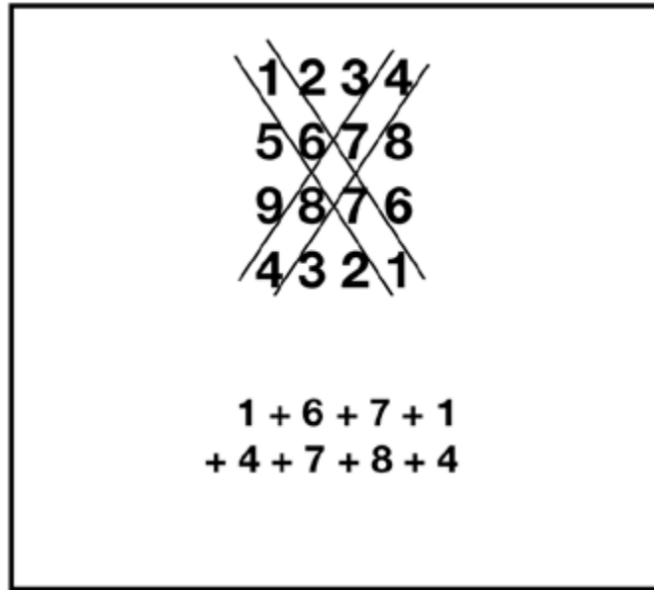
8-38 format

```
1. use std::fmt::{self, Formatter, Display};
2. struct City {
3.     name: &'static str,
4.     lat: f32,
5.     lon: f32,
6. }
7. impl Display for City {
8.     fn fmt(&self, f: &mut Formatter) -> fmt::Result {
9.         let lat_c = if self.lat >= 0.0 { 'N' } else { 'S' };
10.        let lon_c = if self.lon >= 0.0 { 'E' } else { 'W' };
11.        write!(f, "{}: {:.3}°{} {:.3}°{}",
12.            self.name, self.lat.abs(), lat_c, self.lon.abs(), lon_c)
13.    }
14. }
15. fn main() {
16.     let city = City { name: "Beijing", lat: 39.90469, lon: -116.40717 };
17.     assert_eq!(format!("{}", city), "Beijing: 39.905°N 116.407°W");
18.     println!("{}", city);
19. }
```

8-38 City Display trait format
city 17

8.1.8

8-8



8-8

8-39

8-39

```
1. fn main() {
2.     let s = r"1234
3.         5678
4.         9876
5.         4321";
6.     let (mut x, mut y) = (0, 0);
7.     for (idx, val) in s.lines().enumerate() {
8.         let val = val.trim();
9.         let left = val.get(idx..idx+1)
10.            .unwrap().parse::<u32>().unwrap();
11.         let right = val.get((3 - idx)..(3 - idx+1))
12.            .unwrap().parse::<u32>().unwrap();
13.         x += left;
14.         y += right;
15.     }
16.     assert_eq!(38, x+y);
17. }
```



```

    8-39 2 5 r...
    6
    7 15 for
    lines enumerate
    8 for trim
    9 10 get
    parser u32
    11 12 get
    3
    13 14 16
    38
    
```

8.2

Rust

- **Vec**`T`
- **VecDeque**`T` FIFO
- **LinkedList**`T`
- **BinaryHeap**`T`
- **HashMap**`K V` K-V
- **BTreeMap**`K V` B-Tree Key
- **HashSet**`T`
- **BTreeSet**`T` B-Tree

Vec`T` HashMap`K V`

8.2.1 向量的使用

Rust 提供了两种向量的使用方式：一种是使用 `array` 宏，另一种是使用 `Vec` 类型。前者适用于已知大小的静态数组，后者适用于动态增长的向量。例如，以下代码展示了如何使用 `array` 宏和 `Vec` 类型来定义和初始化向量。

```
use std::vec::Vec;
```

```
fn main() {  
    // 使用 array 宏定义静态数组  
    let arr: [i32; 4] = array![1, 2, 3, 4];  
    // 使用 Vec 类型定义动态向量  
    let vec: Vec<i32> = Vec::new();  
    vec.push(1);  
    vec.push(2);  
    vec.push(3);  
    vec.push(4);  
}
```

在代码中，`array` 宏用于定义静态数组，而 `Vec` 类型用于定义动态向量。通过调用 `push` 方法，可以向 `Vec` 中添加元素。

```

1. fn main() {
2.     let mut vec = Vec::new();
3.     vec.push(1);
4.     vec.push(2);
5.     assert_eq!(vec.len(), 2);
6.     assert_eq!(vec[0], 1);
7.     assert_eq!(vec.pop(), Some(2));
8.     assert_eq!(vec.len(), 1);
9.     vec[0] = 7;
10.    assert_eq!(vec[0], 7);
11.    assert_eq!(vec.get(0), Some(&7));
12.    assert_eq!(vec.get(10), None);
13.    // vec[10];
14.    vec.extend([1, 2, 3].iter().cloned());
15.    assert_eq!(vec, [7, 1, 2, 3]);
16.    assert_eq!(vec.get(0..2), Some(&[7, 1][..]));
17.    let mut vec2 = vec![4, 5, 6];
18.    vec.append(&mut vec2);
19.    assert_eq!(vec, [7, 1, 2, 3, 4, 5, 6]);
20.    assert_eq!(vec2, []);
21.    vec.swap(1, 3);
22.    assert!(vec == [7, 3, 2, 1, 4, 5, 6]);
23.    let slice = [1, 2, 3, 4, 5, 6, 7];
24.    vec.copy_from_slice(&slice);
25.    assert_eq!(vec, slice);
26.    let slice = [4, 3, 2, 1];
27.    vec.clone_from_slice(&slice);
28.    assert_eq!(vec, slice);
29. }

```

8-40 Vec new Vector
 String new Rust
 3 4 push i32
 5 len vec 2

```

    6. 测试 pop 方法
    7. 测试 pop 方法，vec 返回的是 Option[T]，如果 vec 为空，则返回 None，否则返回 vec 的最后一个元素。len 方法返回 vec 的长度。
    8. 测试 len 方法，vec 的长度为 10。
    9. 测试 get 方法，vec 的第 13 个元素（索引为 12）不存在，返回 None。
    10. 测试 extend 方法，vec 添加 15 个元素。
    11. 测试 Vector 的 get 方法。
    12. 测试 append 方法，vec 添加 20 个元素。
    13. 测试 swap 方法，vec 的第 21 个元素和第 22 个元素互换。
    14. 测试 copy_from_slice 方法，vec 复制 25 个元素。
    15. 测试 Copy 方法。
    16. 测试 clone_from_slice 方法，vec 复制 27 个元素。
    17. 测试 clone_from_slice 方法，vec 复制 27 个元素。
    18. 测试 with_capacity 方法，vec 的容量为 41。

```

8-41 Vector

```

1. fn main() {
2.     let mut vec = Vec::with_capacity(10);
3.     for i in 0..10 {vec.push(i);}
4.     vec.truncate(0);
5.     assert_eq!(10, vec.capacity());
6.     for i in 0..10 { vec.push(i);}
7.     vec.clear();
8.     assert_eq!(10, vec.capacity());
9.     vec.shrink_to_fit();
10.    assert_eq!(0, vec.capacity());
11.    for i in 0..10 {
12.        vec.push(i);
13.        // output: 4/4/4/4/8/8/8/8/16/16/
14.        print!("{:?}/", vec.capacity());
15.    }
16. }

```

8-41 Vec with_capacity String with_capacity 10

3 for vec 0 9 4

4 truncate 0 clear

6 8 clear

9 shrink_to_fit vec

11 15 vec 0 9 14 4 8 8 16

Vec with_capacity Vector 8-41 i32 4 Rust

ЗСТ Vec new
8-42 Vector

8-42 Vector

```
1. struct Foo;  
2. fn main() {  
3.     let mut vec = Vec::new();  
4.     vec.push(Foo);  
5.     assert_eq!(vec.capacity(), std::usize::MAX);  
6. }
```

8-42 Foo 3
Vec new Vector Vector
String as_ptr
len capacity
Rust

4 push Foo Foo

5 vec std::usize::MAX
Rust

Vector

contains starts_with ends_with
8-43

8-43 contains

```
1. fn main() {
```

```

2.      let v = [10, 40, 30];
3.      assert!(v.contains(&30));
4.      assert!(!v.contains(&50));
5.      assert!(v.starts_with(&[10]));
6.      assert!(v.starts_with(&[10, 40]));
7.      assert!(v.ends_with(&[30]));
8.      assert!(v.ends_with(&[40, 30]));
9.      assert!(v.ends_with(&[]));
10.     let v: &[u8] = &[];
11.     assert!(v.starts_with(&[]));
12.     assert!(v.ends_with(&[]));
13. }

```

8-43 `contains` `starts_with` `ends_with`
`trait PartialEq` `T` `trait`
`contains` `starts_with` `ends_with`

`binary_search`
 8-44

8-44 **binary_search**

```

1. fn main() {
2.     let s = [0, 1, 1, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55];
3.     assert_eq!(s.binary_search(&13), Ok(9));
4.     assert_eq!(s.binary_search(&4), Err(7));
5.     let r = s.binary_search(&1);
6.     assert!(match r { Ok(1...4) => true, _ => false, });
7.     let seek = 13;
8.     assert_eq!(
9.         s.binary_search_by(|probe| probe.cmp(&seek)),
10.        Ok(9)
11.    );
12.    let s = [(0, 0), (2, 1), (4, 1), (5, 1), (3, 1),
13.        (1, 2), (2, 3), (4, 5), (5, 8), (3, 13),
14.        (1, 21), (2, 34), (4, 55)];
15.    assert_eq!(
16.        s.binary_search_by_key(&13, |&(a,b)| b),
17.        Ok(9)
18.    );
19. }

```

`binary_search` `Ord` `Result` `Ok` `Err` `match` `6`

`8-44` `2` `6` `binary_search` `Ord` `Ord` `trait` `binary_search` `&13` `Result` `Ok` `9` `Err` `Result` `match` `6`

`7` `11` `binary_search_by` `FnMut` `&a T` `Ordering` `Ordering` `Less` `Equal` `Greater` `9` `cmp` `Ord` `trait` `Ord` `binary_search_by` `Result`

`12` `18` `binary_search_by_key` `binary_search_by` `binary_search_by_key`

trait

在 Rust 中，trait 是定义在 std 库中的。trait 是 Rust 中定义的一组方法，用于描述一组对象的行为。

在 Rust 中，trait 的定义如下：

- $a \leq a$
- $a \leq b \wedge b \leq a \implies a = b$
- $a \leq b \wedge b \leq c \implies a \leq c$

在 Rust 中，trait 的定义如下：

- $a \leq b \wedge b \leq a \implies a = b$
- $a \leq b \wedge b \leq c \implies a \leq c$
- $a \leq b \wedge b \leq a \implies a = b$

在 Rust 中，trait 的定义如下：

在 Rust 中，trait 的定义如下：

- $a == b$
- $a == b \wedge b == a$
- $a == b \wedge b == c \implies a == c$

在 Rust 中，trait 的定义如下：trait PartialEq<Eq>PartialOrd<Ord>trait

PartialEq<Eq>PartialOrd<Ord>trait

Eq trait PartialEq

- PartialOrd<partial_cmp<lt<le<gt<ge
- Ord<cmp<max<min

Ordering

8-46PartialEqEq

8-46PartialEqEq

```

1. #[lang = "eq"]
2. pub trait PartialEq<Rhs = Self>
3.   where Rhs: ?Sized,
4.   {
5.       fn eq(&self, other: &Rhs) -> bool;
6.       fn ne(&self, other: &Rhs) -> bool { ... }
7.   }
8. pub trait Eq: PartialEq<Self> { }

```

PartialEq 的 eq 和 ne 方法返回 bool 类型的值。PartialEq 和 Eq 是 Rust 中用于比较的 trait。

8-47 PartialOrd

8-47 PartialOrd

```

1. pub trait PartialOrd<Rhs = Self>: PartialEq<Rhs>
2.   where Rhs: ?Sized,
3.   {
4.       fn partial_cmp(&self, other: &Rhs) -> Option<Ordering>;
5.       fn lt(&self, other: &Rhs) -> bool { ... }
6.       fn le(&self, other: &Rhs) -> bool { ... }
7.       fn gt(&self, other: &Rhs) -> bool { ... }
8.       fn ge(&self, other: &Rhs) -> bool { ... }
9.   }
10. pub enum Ordering {
11.     Less,
12.     Equal,
13.     Greater,
14. }

```

8-47 的 std::cmp::PartialOrd trait 定义了 partial_cmp 方法，返回 Option<Ordering> 类型的值。lt, le, gt, ge 方法返回 bool 类型的值。PartialOrd 的 partial_cmp 方法。

8-48 Ord

8-48 Ord

```

1. pub trait Ord: Eq + PartialOrd<Self> {
2.     fn cmp(&self, other: &Self) -> Ordering;
3.     fn max(self, other: Self) -> Self { ... }
4.     fn min(self, other: Self) -> Self { ... }
5. }

```

8-48 `Ord` `Eq` `PartialOrd` `PartialEq` `PartialOrd` `Ord` `cmp` `max` `min` `Ordering` `Option` `Ordering`

Rust `trait` 8-49

8-49

```

1. use std::cmp::Ordering;
2. fn main() {
3.     let result = 1.0.partial_cmp(&2.0);
4.     assert_eq!(result, Some(Ordering::Less));
5.     let result = 1.cmp(&1);
6.     assert_eq!(result, Ordering::Equal);
7.     let result = "abc".partial_cmp(&"Abc");
8.     assert_eq!(result, Some(Ordering::Greater));
9.     let mut v: [f32; 5] = [5.0, 4.1, 1.2, 3.4, 2.5];
10.    v.sort_by(|a, b| a.partial_cmp(b).unwrap());
11.    assert!(v == [1.2, 2.5, 3.4, 4.1, 5.0]);
12.    v.sort_by(|a, b| b.partial_cmp(a).unwrap());
13.    assert!(v == [5.0, 4.1, 3.4, 2.5, 1.2]);
14. }

```

8-49 3 `partial_cmp` `Some` `Ordering` `Less` 4

5 `cmp` `Ordering` `Equal`

7 partial_cmp 8

9 10 sort_by
a.partial_cmp b sort_by a b Less
a b 11 b a 13

trait
[derive]

Vector array
[T] 8-50

8-50 [T]

```

1. pub trait SliceExt {
2.     type Item;
3.     ...
4.     fn split<P>(&self, pred: P) -> Split<Self::Item, P>
5.         where P: FnMut(&Self::Item) -> bool;
6.     ...
7.     fn binary_search(&self, x: &Self::Item) -> Result<usize, usize>
8.         where Self::Item: Ord;
9.     fn len(&self) -> usize;
10.    fn is_empty(&self) -> bool { self.len() == 0 }
11.    fn iter_mut(&mut self) -> IterMut<Self::Item>;
12.    fn swap(&mut self, a: usize, b: usize);
13.    fn contains(&self, x: &Self::Item) -> bool
14.        where Self::Item: PartialEq;
15.    fn clone_from_slice(&mut self, src: &[Self::Item])
16.        where Self::Item: Clone;
17.    fn copy_from_slice(&mut self, src: &[Self::Item])
18.        where Self::Item: Copy;
19.    fn sort_unstable(&mut self) where Self::Item: Ord;
20. }
21. impl<T> SliceExt for [T] {
22.     type Item = T;
23.     ...
24. }

```

8-50 core slice SliceExt trait
 21 [T]
 SliceExt

array array join
 array

Rust 2018 array match match
 array 8-51

8-51 match array

```

1. fn pick(arr: [i32; 3]) {
2.     match arr {
3.         [_, _, 3] => println!("ends with 3"),
4.         [a, 2, c] => println!("{:?}", 2, "{:?}", a, c),
5.         [_, _, _] => println!("pass!"),
6.     }
7. }
8. fn main(){
9.     let arr = [1, 2, 3];
10.    pick(arr);
11.    let arr = [1, 2, 5];
12.    pick(arr);
13.    let arr = [1, 3, 5];
14.    pick(arr);
15. }

```

8-51 pick

Rust array

8-52 match


```

1. use std::collections::HashMap;
2. fn main() {
3.     let mut book_reviews = HashMap::with_capacity(10);
4.     book_reviews.insert("Rust Book", "good");
5.     book_reviews.insert("Programming Rust", "nice");
6.     book_reviews.insert("The Tao of Rust", "deep");
7.     for key in book_reviews.keys() {
8.         println!("{}", key);
9.     }
10.    for val in book_reviews.values() {
11.        println!("{}", val);
12.    }
13.    if !book_reviews.contains_key("rust book") {
14.        println!("find {} times ", book_reviews.len());
15.    }
16.    book_reviews.remove("Rust Book");
17.    let to_find = ["Rust Book", "The Tao of Rust"];
18.    for book in &to_find {
19.        match book_reviews.get(book) {
20.            Some(review) => println!("{}", book, review),
21.            None => println!("{}", book, "is unreviewed."),
22.        }
23.    }
24.    for (book, review) in &book_reviews {
25.        println!("{}", book, review);
26.    }
27.    assert_eq!(book_reviews["The Tao of Rust"], "deep");
28. }

```

8-53 HashMap with_capacity
 HashMap String Vector
 HashMap new
 4 6 insert HashMap
 HashMap HashMap &str &str

0007 12 0000 keys values 0000000000 HashMap 00000000
000000000000 HashMap 0000000000000000000000000000000000
00000000

00013 15 0000 contains_key 0000000000000000000000000000
000000 14 00000000 len 000000 HashMap 00000000

00016 23 0000 remove 000000000000 HashMap 000000000000
HashMap 000000 get 0000000000000000 get 000000 Option T 000000
000000 match 0000000000000000 Some review 0000000000000000
00000000

00024 26 00000000 book review 00 for 000000000000 book 000
review 00

00027 0000 Index 0000000000000000000000000000000000 Rust 00
Index 000000 IndexMut 000000000000 hash[key] 000000000000
hash[key]=value 000
00000000

Entry 00

00 HashMap 000000 Bucket 000000000000 “ ” “ ” 000 Rust 00
000000000000 Entry 000000000000000000000000 8-54 000

0000 8-54 Entry 00

1. pub enum Entry<'a, K: 'a, V: 'a> {
2. Occupied(OccupiedEntry<'a, K, V>),
3. Vacant(VacantEntry<'a, K, V>),
4. }

00000 8-54 0000 Entry 0000000000000000000000000000000000 Occupied
OccupiedEntry 00 a K V 0000 Vacant VacantEntry 00 a K V
0000 OccupiedEntry 00 a K V 0000 VacantEntry 00 a K V 0000000000
000000000000 HashMap 00
00

Entry 000 HashMap 0000000000000000
000 8-55 000

0000 8-55 entry 00000000

```

1. use std::collections::HashMap;
2. fn main() {
3.     let mut map: HashMap<&str, u32> = HashMap::new();
4.     map.entry("current_year").or_insert(2017);
5.     assert_eq!(map["current_year"], 2017);
6.     *map.entry("current_year").or_insert(2017) += 10;
7.     assert_eq!(map["current_year"], 2027);
8.     let last_leap_year = 2016;
9.     map.entry("next_leap_year")
10.        .or_insert_with(|| last_leap_year + 4 );
11.     assert_eq!(map["next_leap_year"], 2020);
12.     assert_eq!(map.entry("current_year").key(), &"current_year");
13. }

```

8-55 Entry or_insert or_insert_with key entry Entry K V

3 HashMap new HashMap 4 entry Entry "current_year" entry hash hash hash Occupied Vacant Entry K V

Entry or_insert or_insert_with 6 "*" or_insert

8 10 or_insert_with FnOnce V

12 key Entry

8-56 or_insert

8-56 or_insert

```

1. pub fn or_insert(self, default: V) -> &'a mut V {
2.     match self {
3.         Occupied(entry) => entry.into_mut(),
4.         Vacant(entry)   => entry.insert(default),
5.     }
6. }

```

8-56 `entry` `match` `Occupied` `entry` `into_mut` `Vacant` `entry` `insert` `insert` `VacantEntry` `insert`

HashMap

HashMap 8-57

8-57 HashMap

```

1. use std::collections::HashMap;
2. fn merge_extend<'a>(
3.     map1: &mut HashMap<'a str, 'a str>,
4.     map2: HashMap<'a str, 'a str>
5. ) {
6.     map1.extend(map2);
7. }
8. fn merge_chain<'a>(
9.     map1: HashMap<'a str, 'a str>,
10.    map2: HashMap<'a str, 'a str>
11. ) -> HashMap<'a str, 'a str> {
12.    map1.into_iter().chain(map2).collect()
13. }

```

```

14. fn merge_by_ref<'a>(
15.     map: &mut HashMap<'a str, &'a str>,
16.     map_ref: &HashMap<'a str, &'a str>
17. ){
18.     map.extend(map_ref.into_iter()
19.                 .map(|(k, v)| (k.clone(), v.clone())))
20. };
21. }
22. fn main() {
23.     let mut book_reviews1 = HashMap::new();
24.     book_reviews1.insert("Rust Book", "good");
25.     book_reviews1.insert("Programming Rust", "nice");
26.     book_reviews1.insert("The Tao of Rust", "deep");
27.     let mut book_reviews2 = HashMap::new();
28.     book_reviews2.insert("Rust in Action", "good");
29.     book_reviews2.insert("Rust Primer", "nice");
30.     book_reviews2.insert("Matering Rust", "deep");
31.     // merge_extend(&mut book_reviews1, book_reviews2);
32.     // let book_reviews1 = merge_chain(book_reviews1, book_reviews2);
33.     merge_by_ref(&mut book_reviews1, &book_reviews2);
34.     for key in book_reviews1.keys() {
35.         println!("{}", key);
36.     }
37. }

```

8-57 HashMap

2 7 merge_extend extend HashMap 31 extend HashMap

8 13 merge_chain into_iter Chain 32

14 21 merge_by_ref extend HashMap 33 HashMap 34 for

HashMap

HashMap

1 Hash

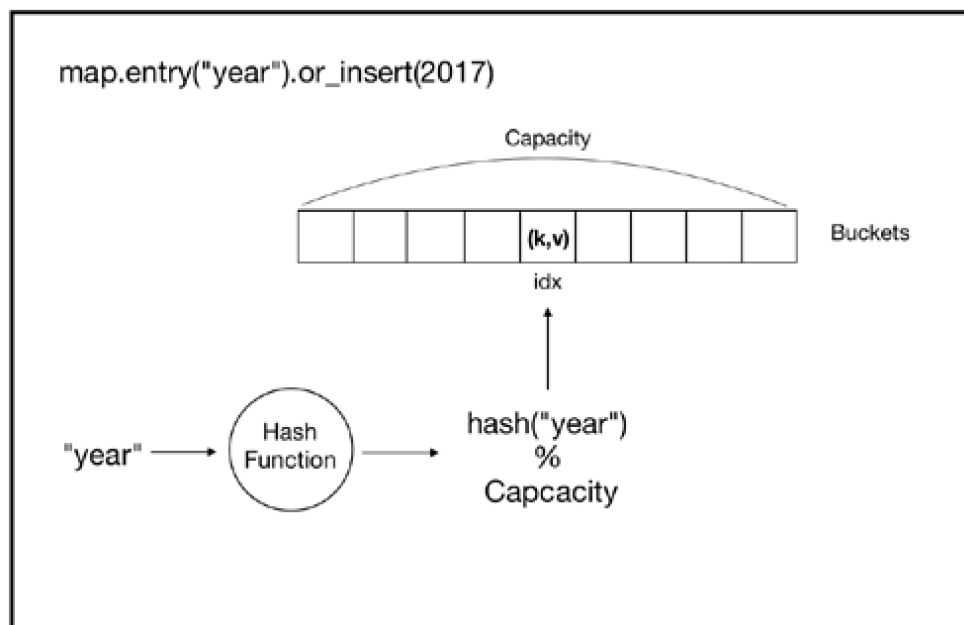
2 Hash

3 HashMap

HashMap

Hash Hash Capacity

HashMap 8-9



8-9 HashMap

HashMap Hash

Hash Hash

Hash

Hash Hash

Hash Hash Collision Hash

Hash

Hash Load Factor

100 90 0.9 HashMap

HashMap 使用 SipHash 13 和 SipHash 24 的 Hash 函数，可以防止 Hash 碰撞攻击。

Rust 的 HashMap 使用 SipHash 13 和 SipHash 24 的 Hash 函数，可以防止 Hash 碰撞攻击。Hash Collision DoS 攻击是指攻击者通过构造大量的 Hash 值，使得 CPU 的 Hash 计算达到 100% 的利用率，从而导致系统崩溃。SipHash 是一种快速的 Hash 函数，Rust 使用 SipHash 13 和 SipHash 24 的 Hash 函数，可以防止 Hash 碰撞攻击。FNV 是一种快速的 Hash 函数，Rust 使用 FNV 的 Hash 函数，可以防止 Hash 碰撞攻击。

8-58 Rust Hash trait

8-58 Rust Hash trait

```
1. pub trait Hasher {
2.     fn finish(&self) -> u64;
3.     fn write(&mut self, bytes: &[u8]);
4. }
5. pub trait Hash {
6.     fn hash<H>(&self, state: &mut H) where H: Hasher;
7. }
8. pub trait BuildHasher {
9.     type Hasher: Hasher;
10.    fn build_hasher(&self) -> Self::Hasher;
11. }
```

8-58 Rust Hash trait 14 Hasher Hash write finish

5 7 Hash trait hash Hasher write

8 11 BuildHasher Hasher build_hasher Hasher

trait Rust Hash Rust Hash Eq trait HashMap HashMap

8-59 fnv Fnv

8-59 Fnü

```
1. pub struct FnvHasher(u64);
2. impl Hasher for FnvHasher {
3.     fn finish(&self) -> u64 {
4.         self.0
5.     }
6.     fn write(&mut self, bytes: &[u8]) {
7.         let FnvHasher(mut hash) = *self;
8.         for byte in bytes.iter() {
9.             hash = hash ^ (*byte as u64);
10.            hash = hash.wrapping_mul(0x100000001b3);
11.        }
12.        *self = FnvHasher(hash);
13.    }
14. }
```

8-59 Hasher

HashMap 8-60

8-60 HashMap

```
1. pub struct HashMap<K, V, S = RandomState> {
2.     hash_builder: S,
3.     table: RawTable<K, V>,
4.     resize_policy: DefaultResizePolicy,
5. }
6. impl<K: Hash + Eq, V> HashMap<K, V, RandomState> {
7.     pub fn new() -> HashMap<K, V, RandomState> {
8.         Default::default()
9.     }
10. }
11. impl<K, V, S> Default for HashMap<K, V, S>
12.     where K: Eq + Hash, S: BuildHasher + Default
```



```

13.  {
14.    fn default() -> HashMap<K, V, S> {
15.      HashMap::with_hasher(Default::default())
16.    }
17. }

```

8-60 15 HashMap hash_builder table resize_policy hash_builder S RandomState RandomState DefaultHasher SipHasher13 Hash RandomState Hash

table HashMap RawTable K V resize_policy HashMap 0.9

6 10 HashMap new Default default 11 17 HashMap Default trait with_hasher SipHasher13

Hash Hash Hash Hash

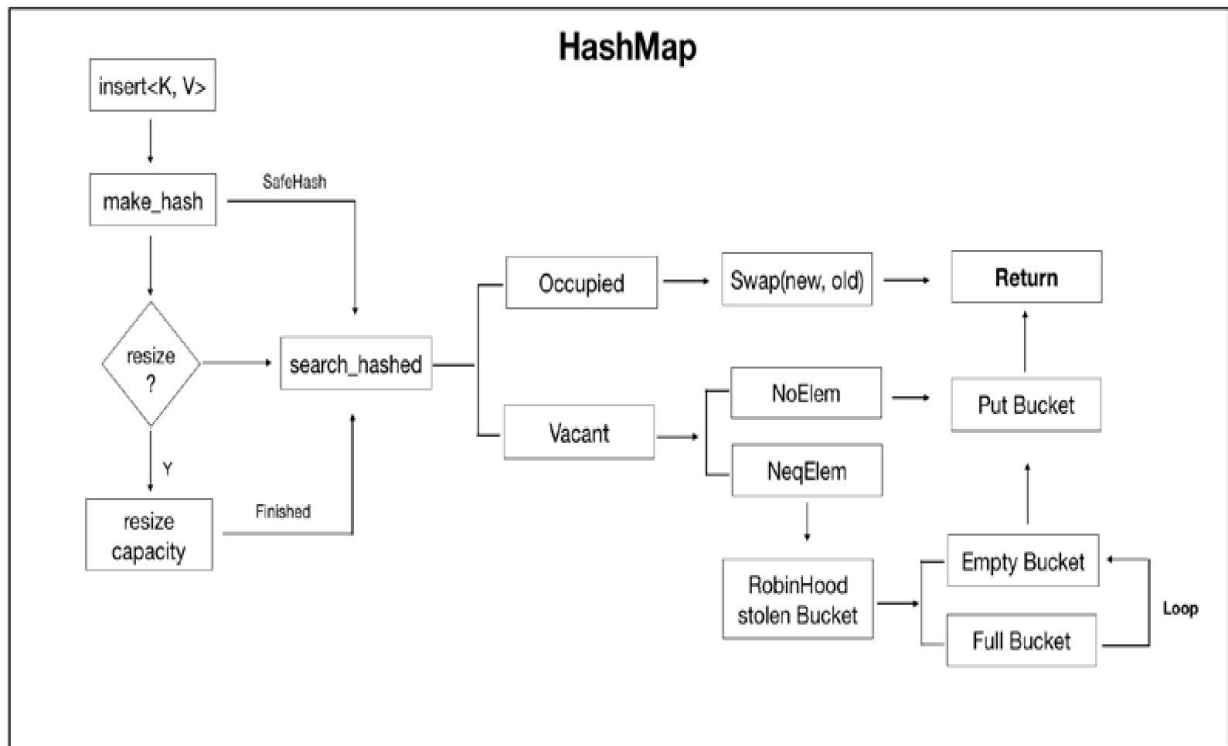
Bucket Hash HashMap

Probe Linear Probing Quadratic Probing Ruby 2.4 Ruby Python

Hash Hash

Rust 的 HashMap 使用 **Robin Hood Hashing** 算法。该算法在插入元素时，如果当前桶已满，则会将桶中的元素移动到下一个桶中，直到找到空桶为止。这种算法可以保证元素的插入顺序与哈希值的顺序一致，从而避免了传统哈希表中的“二次哈希”问题。

8-10 Rust 的 HashMap 内部结构



8-10 Rust 的 HashMap 内部结构

HashMap 的 insert 方法首先调用 make_hash 生成哈希值。如果哈希值已经存在，则调用 SafeHash 进行二次哈希。如果哈希值不存在，则调用 resize 检查是否需要扩容。如果不需要扩容，则调用 search_hashed 查找元素。

search_hashed 方法在 HashMap 的 table 中查找元素。如果找到元素，则调用 SafeHash 进行二次哈希。如果未找到元素，则调用 NoElem 或 NeqElem 处理。NeqElem 会调用 RobinHood 算法，将元素移动到下一个桶中，直到找到空桶为止。如果找到空桶，则调用 Empty Bucket 处理。如果桶已满，则调用 Full Bucket 处理，并进入 Loop 循环。

8-61 VacentEntryState

```

1. enum VacantEntryState<K, V, M> {
2.     NeqElem(FullBucket<K, V, M>, usize),
3.     NoElem(EmptyBucket<K, V, M>, usize),
4. }

```

8-61 8-61 8-61 NeqElem FullBucket NoElem EmptyBucket VacantEntryState Vacant Rust Enum

EmptyBucket NoElem NoElem insert FullBucket NeqElem

NeqElem FullBucket Rust robin_hood robin_hood loop

FullBucket Hash search_hashed Hash Occupied Occupied insert std mem swap get

HashMap Rust

HashMap remove Hash Hash search_hashed None pop_internal gap_peek GapThenFull 8-62

8-62 GapThenFull

```

1. pub struct GapThenFull<K, V, M> {
2.     gap: EmptyBucket<K, V, ()>,
3.     full: FullBucket<K, V, M>,
4. }

```

GapThenFull remove Enum

HashMap 的容量是动态调整的，当 HashMap 的容量达到一定程度时，HashMap 会自动调用 extend 方法，将容量扩大一倍，for 循环遍历 HashMap 中的元素。

8.3 Vec

Vec 是 Rust 中的动态数组，它类似于 C++ 中的 vector。Vec 的容量（Capacity）和长度（Len）是两个重要的属性。图 8-11 展示了 Vec 的内部结构。

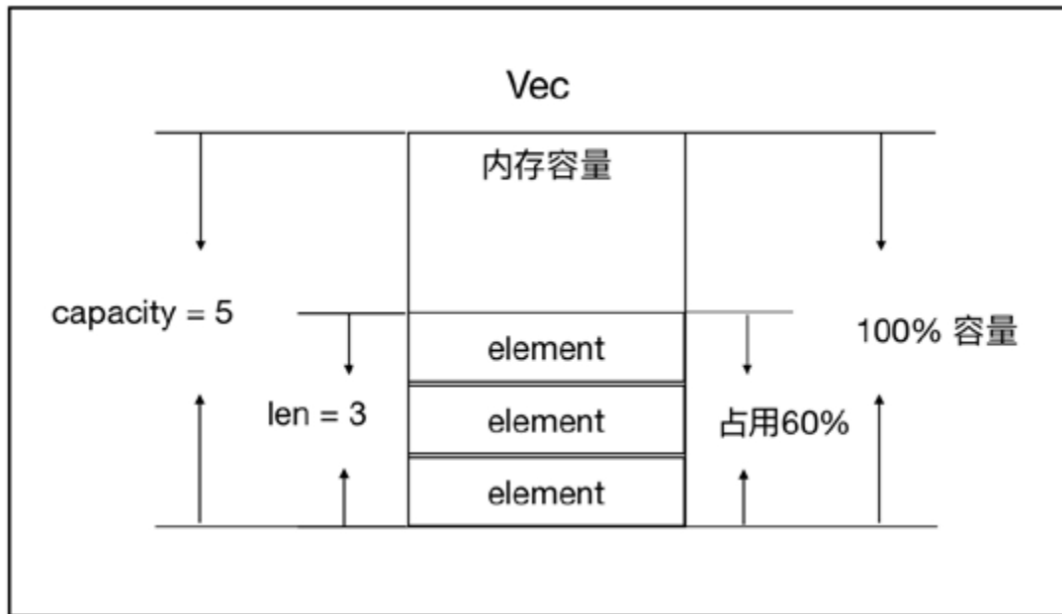


图 8-11 Vec 的内部结构

图 8-11 展示了 Vec 的内部结构。Vec 的容量（Capacity）和长度（Len）是两个重要的属性。Vec 的容量是动态调整的，当 Vec 的容量达到一定程度时，Vec 会自动调用 reserve 方法，将容量扩大一倍。Vec 的长度是动态调整的，当 Vec 的长度达到一定程度时，Vec 会自动调用 push 方法，将元素添加到 Vec 的末尾。Vec 的内部结构是一个数组，每个元素都是一个 T 类型的值。Vec 的容量是 5，长度是 3，占用了 60% 的容量。

Rust 1.21 到 1.3 版本中，VecDeque 的 reserve 方法被弃用。在 Rust 1.21 之前，VecDeque 的 reserve 方法是存在的，但在 Rust 1.21 之后，它被移到了 VecDeque 的 trait 方法中。因此，在 Rust 1.21 及以后版本中，应该使用 VecDeque::reserve 方法来扩容 VecDeque。

Rust 中的 **VecDeque** 是一个双端队列（Double-Ended Queue），它支持在两端进行插入和删除操作。VecDeque 的容量是动态调整的，当 VecDeque 的容量达到一定程度时，VecDeque 会自动调用 reserve 方法，将容量扩大一倍。VecDeque 的内部结构是一个 Ring Buffer，它是一个环形缓冲区，可以高效地进行插入和删除操作。图 8-12 展示了 VecDeque 的内部结构。

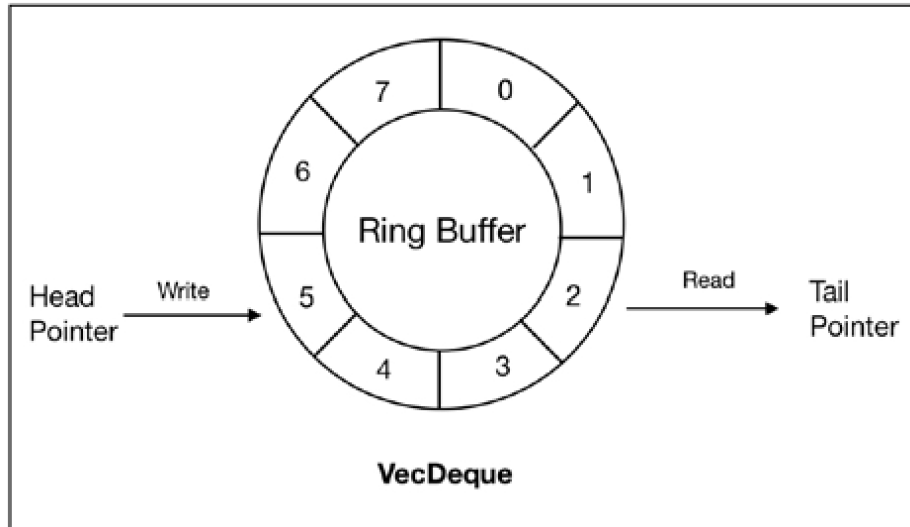


图8-12 VecDeque 的内部结构

VecDeque 的内部结构是一个环形缓冲区（Ring Buffer）。它由一个数组（Array）组成，数组的大小为 `2 * capacity`。数组的头部（Head）和尾部（Tail）指针分别指向当前写入和读取的位置。Head 指针从 0 开始，随着写入操作向右移动；Tail 指针从 0 开始，随着读取操作向右移动。当 Head 指针到达数组的末尾时，它会绕回到数组的开头，从而实现循环利用。

图8-12展示了 VecDeque 的内部结构。图中显示了一个容量为 8 的 VecDeque，其内部数组的大小为 16。Head 指针指向索引 1，Tail 指针指向索引 0。数组中的元素按照索引顺序排列，从 0 到 7。Head 指针和 Tail 指针的移动方向分别由 'Write' 和 'Read' 箭头指示。当 Head 指针到达数组的末尾时，它会绕回到数组的开头，从而实现循环利用。

Rust 的 VecDeque 实现使用了上述的环形缓冲区结构。在 Rust 中，VecDeque 的容量（Capacity）可以通过 `with_capacity` 方法指定。图8-63展示了 VecDeque 的容量和元素的关系。

图8-63展示了 VecDeque 的容量和元素的关系。图中显示了一个容量为 8 的 VecDeque，其内部数组的大小为 16。Head 指针指向索引 1，Tail 指针指向索引 0。数组中的元素按照索引顺序排列，从 0 到 7。Head 指针和 Tail 指针的移动方向分别由 'Write' 和 'Read' 箭头指示。当 Head 指针到达数组的末尾时，它会绕回到数组的开头，从而实现循环利用。

```

1. pub fn with_capacity(n: usize) -> VecDeque<T> {
2.     // 此处使用+1, 是因为 ringbuffer 总是需要预留一个空位
3.     let cap = cmp::max(n + 1, MINIMUM_CAPACITY + 1)
4.         .next_power_of_two();
5.     assert!(cap > n, "capacity overflow");
6.     VecDeque {
7.         tail: 0,
8.         head: 0,
9.         buf: RawVec::with_capacity(cap),
10.    }
11. }

```

图8-63 `VecDeque<T>`的 `with_capacity` 函数实现

图8-63展示了 `with_capacity` 函数的实现。该函数接收一个参数 `n`，并返回一个容量至少为 `n+1` 的 `VecDeque`。代码中使用了 `cmp::max` 和 `next_power_of_two` 来确保容量是2的幂次，并且至少为 `MINIMUM_CAPACITY + 1`。同时，使用 `assert!` 来保证容量大于 `n`，防止溢出。

图8-64展示了 `VecDeque` 的 `is_full` 函数实现。

图8-64 `VecDeque<T>`的 `is_full` 函数实现

```

1. fn is_full(&self) -> bool {
2.     self.cap() - self.len() == 1
3. }

```

图8-64展示了 `VecDeque` 的 `is_full` 函数实现。该函数检查当前容量减去当前长度是否等于1，如果是，则返回 `true`，表示队列已满。

图8-65展示了 `VecDeque` 的 `is_full` 函数实现。

图8-65 `VecDeque<T>`的 `is_full` 函数实现

```

1. pub fn reserve(&mut self, additional: usize) {
2.     let old_cap = self.cap();
3.     let used_cap = self.len() + 1;
4.     let new_cap = used_cap.checked_add(additional)
5.         .and_then(
6.             |needed_cap| needed_cap.checked_next_power_of_two()
7.         ).expect("capacity overflow");
8.     if new_cap > self.capacity() { // 问题代码
9.         self.buf.reserve_exact(used_cap, new_cap - used_cap);
10.        unsafe {
11.            self.handle_cap_increase(old_cap);
12.        }
13.    }
14. }

```

8-65 reserve 8

8 capacity VecDeque T capacity cap cap=capacity+1 capacity cap

8 "if new_cap old_cap{ " old_cap

" " " " "

8.4

Rust &str String &str

`&str` 是 `String` 的引用，`String` 是字符串的堆内存地址。

`Rust` 使用 `UTF-8` 编码，`String` 是 `Vec<u8>` 的别名。

`Pattern` 是正则表达式，`a` 是正则表达式，`contains` 是 `trim` 方法，`trim_matches` 是 `find` 方法，`Rust` 使用 `regex` 库。

`&str` 是 `String` 的引用，`DST` 是 `&str` 的别名，`String` 是 `Vec<u8>` 的别名，`array` 是 `Vector` 的别名，`slice` 是 `with_capacity` 方法，`String` 是 `Vector` 的别名，`Index` 是 `Rust` 的别名。

`HashMap` 是 `Rust` 的别名，`Eq` 是 `Hash` 的别名，`HashMap` 是 `HashMap` 的别名，`NewType` 是 `HashMap` 的别名。

`HashMap` 是 `Index` 的别名，`IndexMut` 是 `Index` 的别名，`Rust` 是 `Entry` 的别名，`HashMap` 是 `extend` 的别名，`for` 是 `HashMap` 的别名。

`HashMap` 是 `Rust` 的别名，`Hash` 是 `HashMap` 的别名，`0.9` 是 `HashMap` 的别名，`HashMap` 是 `Enum` 的别名，`remove` 是 `HashMap` 的别名。

`VecDeque` 是 `T` 的别名，`reserve` 是 `CVE` 的别名，`“”` 是 `Rust` 的别名，`Rust` 是 `Rust` 的别名。

`std` 是 `collection` 的别名，`LinkedList` 是 `T` 的别名，`BinaryHeap` 是 `T` 的别名，`BTreeMap` 是 `T` 的别名，`HashSet` 是 `T` 的别名，`BTreeSet` 是 `T` 的别名，`Rust` 是 `Rust` 的别名，`Vec` 是 `T` 的别名，`HashMap` 是 `K` 的别名，`V` 是 `V` 的别名。

9 异常处理

异常处理是程序设计中不可或缺的一部分。

异常处理允许程序在遇到错误或异常情况时，能够以一种可控的方式进行处理，而不是崩溃或进入未知状态。这有助于提高程序的健壮性和可维护性。

异常处理的核心概念是 **Robust**（健壮性）。一个健壮的程序能够处理各种可能的异常情况，并在出现问题时提供有意义的反馈。这通常涉及到捕获异常、记录日志以及采取适当的恢复措施。

在异常处理中，我们通常会使用一些关键字或函数来定义和抛出异常。例如，在 C 语言中，我们可能会使用 `goto` 或 `setjump` 来实现非局部跳转。

在 C 语言中，我们可能会使用 `assert` 宏来检查程序中的断言。如果断言失败，程序会调用 `abort` 函数并终止。在 Java 中，我们可能会使用 `Bug` 类来记录程序中的错误。在 Rust 中，我们可能会使用 `“panic!”` 来引发异常。

C++ 和 Java 等语言都提供了异常处理机制。在 C++ 中，我们可能会使用 `Stack Unwind` 来处理异常。在 Java 中，我们可能会使用 `Stack Backtrack` 来处理异常。这些机制允许程序在遇到异常时，能够回溯到异常发生的位置，并进行相应的处理。

Rust 语言也提供了异常处理机制。在 Rust 中，我们可能会使用 `Failure` 或 `Error` 类型来处理异常。这些类型通常用于表示程序中的失败状态，并允许程序在遇到异常时，能够以一种可控的方式进行处理。

9.1 异常类型

异常类型是指程序在遇到异常时所返回的值的类型。常见的异常类型包括 `Failure`、`Error` 和 `Exception`。这些类型通常用于表示程序中的失败状态，并允许程序在遇到异常时，能够以一种可控的方式进行处理。


```

1. fn sum(a: i32, b: i32) -> i32 {
2.     a + b
3. }
4. fn main() {
5.     sum(1u32, 2u32);
6. }

```

9-1 `sum` `i32` `main` `u32` “”

error[E0308]: mismatched types

```

5 |     sum(1u32, 2u32);
  |           ^^^^ expected i32, found u32

```

`sum` `i32` `u32`

`Vector` `insert` 9-2

9-2 `Vec` `T` `insert`

```

1. fn main() {
2.     let mut vec = vec![1, 2, 3];
3.     vec.insert(1, 4);
4.     assert_eq!(vec, [1, 4, 2, 3]);
5.     vec.insert(4, 5);
6.     assert_eq!(vec, [1, 4, 2, 3, 5]);
7.     // vec.insert(8, 8);
8. }

```

9-2 `insert` 3 5 7

7 `Assert` `Rust`

- `assert` `true`
- `assert_eq` `PartialEq`
- `assert_ne` `PartialEq`

- `debug_assert` `assert`
- `debug_assert_eq` `assert_eq`
- `debug_assert_ne` `assert_ne`

`assert` `Debug` `Release`
`debug_assert`
`assert` `debug_assert`

9-2 `insert` `assert`
 9-3

9-3 `insert` `assert`

```
1. pub fn insert(&mut self, index: usize, element: T) {
2.     let len = self.len();
3.     assert!(index <= len);
4.     ...
5. }
```

9-3 `Vec` `T` `insert` 3 `assert`
`index` `len` `len`
`false` `assert`

Fast Fail
`Bug` `assert` 9-4

9-4

```
1. fn main() {
2.     let x = false;
3.     assert!(x, "x wasn't true!");
4.     let a = 3; let b = 28;
5.     debug_assert!(a + b == 30, "a = {}, b = {}", a, b);
6. }
```

9-4 3 5

“”
 “” `Bug`

- [illegible]

```
panic assert
```

Rust

- **Option T** `HashMap`

- ## Result TE

- Panic**

- **Abort** `abort`

Rust Haskell

9.3.1 Option T

Option T ☐ Some T ☒ None ☐

Option T 9-5

9-5 Option T

```

1. fn get_shortest(names: Vec<&str>) -> Option<&str> {
2.     if names.len() > 0 {
3.         let mut shortest = names[0];
4.         for name in names.iter() {
5.             if name.len() < shortest.len() {
6.                 shortest = *name;
7.             }
8.         }
9.         Some(shortest)
10.    } else {
11.        None
12.    }
13. }
14. fn show_shortest(names: Vec<&str>) -> &str {
15.    match get_shortest(names) {
16.        Some(shortest) => shortest,
17.        None           => "Not Found",
18.    }
19. }
20. fn main(){
21.    assert_eq!(show_shortest(vec!["Uku", "Felipe"]), "Uku");
22.    assert_eq!(show_shortest(Vec::new()), "Not Found");
23. }

```

9-5 `get_shortest` `Vec<&str>` `Option<&str>`
`Vec::new()` `Option::None` `Some`
`get_shortest` `Option<&str>`

14-19 `show_shortest` `get_shortest`
`match` `Option<&str>` `Some`
`None` `"Not Found"` `main` `show_shortest`

unwrap

9-5 Option T match
unwrap 9-6

9-6 unwrap

```
1. fn show_shortest(names: Vec<&str>) -> &str {
2.     // get_shortest(names).unwrap()
3.     get_shortest(names).unwrap_or("Not Found")
4.     // get_shortest(names).unwrap_or_else(|| "Not Found")
5.     // get_shortest(names).expect("Not Found")
6. }
7. fn main(){
8.     assert_eq!(show_shortest(vec!["Uku", "Felipe"]), "Uku");
9.     assert_eq!(show_shortest(Vec::new()), "Not Found");
10. }
```

9-6 unwrap unwrap_or unwrap_or_else
unwrap Some None
show_shortest 2

3 unwrap_or match
None "Not Found" 9-5

4 unwrap_or_else unwrap_or
FnOnce T

5 expect None

unwrap unwrap
Bug expect None
match unwrap_or unwrap_or_else

Option T

Option T
Option T unwrap
None match 9-7

9-7 match Option T


```

1. fn get_shortest_length(names: Vec<&str>) -> Option<usize> {
2.     match get_shortest(names) {
3.         Some(shortest) => Some(shortest.len()),
4.         None           => None,
5.     }
6. }
7. fn main(){
8.     assert_eq!(get_shortest_length(vec!["Uku","Felipe"]),Some(3));
9.     assert_eq!(get_shortest_length(Vec::new()), None);
10. }

```

9-7 `get_shortest_length` `Vec<&str>` `Option<usize>`
 2 `match` `get_shortest` `Option<&str>`
`Some` `shortest` `len` `Some`
`None` `None` `match`
`std::option::map` 9-8

9-8 `map` `Option<T>`

```

1. fn get_shortest_length(names: Vec<&str>) -> Option<usize> {
2.     get_shortest(names).map(|name| name.len())
3. }
4. fn main(){
5.     assert_eq!(get_shortest_length(vec!["Uku","Felipe"]),Some(3));
6.     assert_eq!(get_shortest_length(Vec::new()), None);
7. }

```

9-8 `map` 9-7
`map` `match` 9-9

9-9 `std::option::Option` `map`

```

1. pub fn map<U, F: FnOnce(T) -> U>(self, f: F) -> Option<U> {
2.     match self {
3.         Some(x) => Some(f(x)),
4.         None   => None,
5.     }
6. }

```

9-9 map match Some None FnOnce T-U map Option T Option T map Combinator

map map_or map_or_else map match None unwrap_or unwrap_or_else

map Option T T Option T map Some inverse double log 2 square sqrt NaN Option T 9-10

9-10 map and_then

```
1. fn double(value: f64) -> f64 {
2.     value * 2.
3. }
4. fn square(value: f64) -> f64 {
5.     value.powi(2 as i32)
6. }
7. fn inverse(value: f64) -> f64 {
8.     value * -1.
9. }
10. fn log(value: f64) -> Option<f64> {
11.     match value.log2() {
```

```

12.         x if x.is_normal() => Some(x),
13.         _                    => None
14.     }
15. }
16. fn sqrt(value: f64) -> Option<f64> {
17.     match value.sqrt() {
18.         x if x.is_normal() => Some(x),
19.         _                    => None
20.     }
21. }
22. fn main () {
23.     let number: f64 = 20.;
24.     let result = Option::from(number)
25.         .map(inverse).map(double).map(inverse)
26.         .and_then(log).map(square).and_then(sqrt);
27.     match result {
28.         Some(x) => println!("Result was {}. ", x),
29.         None   => println!("This failed.")
30.     }
31. }

```

9-10 1 9 double square inverse
 f64

10 21 log sqrt Option f64
 NaN log2 sqrt
 is_normal Some None

main number Option from
 Some number 25 map double
 square inverse Some number number
 map map map
 Some Some Some number
 Some Some Some number
 and_then

26 and_then log sqrt map
23 number "This failed."

9-11 and_then

9-11 and_then

```
1. pub fn and_then<U, F>(self, f: F) -> Option<U>
2.     where F: FnOnce(T) -> Option<U>
3.     {
4.         match self {
5.             Some(x) => f(x),
6.             None => None,
7.         }
8.     }
```

9-11 and_then map 5 Some
and_then map Some

map and_then
Option T

9.3.2 Result T E

Option T
Option T Rust Result T E
Option T Result T E
Option T Result T
Option T Result T

9-12 Result T E

9-12 Result T E

```
1. #[must_use]
2. pub enum Result<T, E> {
3.     Ok(T),
4.     Err(E),
5. }
```

9-12 `Result` `T` `E` `Ok` `T` `Err`
`E` `Ok` `T` `Err` `E`
[must_use] `Result` `T` `E`

9-13 `parse`

9-13 `parse`

```
1. fn main() {
2.     let n = "1";
3.     assert_eq!(n.parse::<i32>(), Ok(1));
4.     let n = "a";
5.     // 输出 Err(ParseIntError { kind: InvalidDigit })
6.     println!("{:?}", n.parse::<i32>());
7. }
```

9-13 2 3
4
6 `parse` **Err**
`ParseIntError{kind: InvalidDigit}`
`InvalidDigit`

`Result` `T` `E`

`std::result::Result` `T` `E` `unwrap`
9-13 `unwrap_or`
`std::result::Result` `map` `and_then`
`Option` `T` 9-14 **9-14**

```

1. use std::num::ParseIntError;
2. fn square(number_str: &str) -> Result<i32, ParseIntError>
3. {
4.     number_str.parse::<i32>().map(|n| n.pow(2))
5. }
6. fn main() {
7.     match square("10") {
8.         Ok(n) => assert_eq!(n, 100),
9.         Err(err) => println!("Error: {:?}", err),
10.    }
11. }

```

9-14 square parse i32
 map parse Result
 map square Result i32 ParseIntError
 ParseIntError std num use

main match square Ok n
 Err err

type 9-15

9-15 type

```

1. type ParseResult<T> = Result<T, ParseIntError>;
2. fn square(number_str: &str) -> ParseResult<i32>
3. {
4.     number_str.parse::<i32>().map(|n| n.pow(2))
5. }

```

9-15 1 type Result T ParseIntError
 ParseResult T square

8

8 parse
 FromStr from_str Result F F as FromStr
 Err u32 FromStr from_str

· ParseIntError { kind
 Empty }


```

1. pub struct Error {
2.     repr: Repr,
3. }
4. enum Repr {
5.     Os(i32),
6.     Simple(ErrorKind),
7.     Custom(Box<Custom>),
8. }
9. struct Custom {
10.    kind: ErrorKind,
11.    error: Box<error::Error+Send+Sync>,
12. }
13. pub enum ErrorKind {
14.    NotFound,
15.    PermissionDenied,
16.    ConnectionRefused,
17.    ConnectionReset,
18.    ConnectionAborted,
19.    NotConnected,
20.    ...
21. }

```

9-17 std::io::Error 的 repr 是 Repr 枚举。Repr 枚举包含 Os(i32)、Simple(ErrorKind)、Custom(Box<Custom>) 三个变体。Custom 变体包含一个 ErrorKind 枚举和一个 Box<error::Error+Send+Sync> 类型的错误信息。

9-18 9-17 中的 Error 结构体的 repr 枚举。

9-18 9-17 中的 Error 结构体的 repr 枚举。

9-18 9-17 中的 Error 结构体的 repr 枚举。


```

1. use std::env;
2. use std::fs::File;
3. use std::io::prelude::*;
4. fn main() {
5.     let args: Vec<String> = env::args().collect();
6.     println!("{:?}", args);
7.     let filename = &args[1];
8.     let mut f = File::open(filename).unwrap();
9.     let mut contents = String::new();
10.    f.read_to_string(&mut contents).unwrap();
11.    let mut sum = 0;
12.    for c in contents.lines(){
13.        let n = c.parse::<i32>().unwrap();
14.        sum += n;
15.    }
16.    println!("{:?}", sum);
17. }

```

9-18 `std::env::args()` returns a `Vec<String>` of 5 arguments, the first is `args[0]` which is the program name.

8 `File::open()` returns a `Result<File, Error>` which is a `std::io::Result<File>` type. `unwrap()` returns a `Result<File>` which is a `std::io::Result<File>` type.

9-10 `read_to_string()` returns a `Result<String>` which is a `std::io::Result<String>` type. `UTF-8` encoding is used.

11-15 `lines()` returns an iterator over `String` slices.

`io_origin.rs` `rustc`

```
$ ./io_origin test_txt
["./io_origin", "test_txt"]
In file test_txt
6
```

io_origin test_txt
1 2 3 6

test_txt

thread 'main' panicked at 'called `Result::unwrap()` on an `Err` value: ParseIntError { kind: InvalidDigit }', src/libcore/result.rs:906:4

ParseIntError {kind: InvalidDigit}
9-18

I/O parse Rust
Error trait trait trait
9-19 Error trait

9-19 Error trait

```
1. pub trait Error: Debug + Display {
2.     fn description(&self) -> &str;
3.     fn cause(&self) -> Option<&Error> { ... }
4. }
5. impl<'a, E: Error + 'a> From<E> for Box<Error + 'a> {
6.     fn from(err: E) -> Box<Error + 'a> {
7.         Box::new(err)
8.     }
9. }
```

9-19 description cause
trait Debug Display
Box Error &Error

9-19 Box Error + 'a From trait
From from Error Box Error

9-18
Rust 2015 main Rust 2018 main

9-20

9-20 9-18 run

```
1. use std::env;
2. use std::error::Error;
3. use std::fs::File;
4. use std::io::prelude::*;
5. use std::process;
6. type ParseResult<i32> = Result<i32, Box<Error>>;
7. fn main() {
8.     let args: Vec<String> = env::args().collect();
9.     let filename = &args[1];
10.    println!("In file {}", filename);
11.    match run(filename) {
12.        Ok(n) => { println!("{:?}", n); },
13.        Err(e) => {
14.            println!("main error: {}", e);
15.            process::exit(1);
16.        }
17.    }
18. }
```

9-20 main run
run Result<i32> Box<Error> main match
Err e 1

9-21 run

9-21 run

```

1. fn run(filename: &str) -> ParseResult<i32> {
2.     File::open(filename).map_err(|e| e.into())
3.     .and_then(|mut f|{
4.         let mut contents = String::new();
5.         f.read_to_string(&mut contents)
6.         .map_err(|e| e.into()).map(|_|contents)
7.     })
8.     .and_then(|contents|{
9.         let mut sum = 0;
10.        for c in contents.lines(){
11.            match c.parse::<i32>() {
12.                Ok(n) => {sum += n;},
13.                Err(err) => {
14.                    let err: Box<Error> = err.into();
15.                    println!("error info: {}, cause: {:?}",
16.                        , err.description(),err.cause());
17.                },
18.                // Err(err) => { return Err(From::from(err));},
19.            }
20.        }
21.        Ok(sum)
22.    })
23. }

```

9-21 run Result<i32>Box<Error>
ParseResult<i32>

2 File open Result<File>
map_err
|e|e.into
Box<Error>
Error trait
From trait
Box<Error>
File open return

```
    3 7  and_then  map_err  \
Result::File(4 5  read_to_string  \
  map_err  map  \
  contents
```

```
    8 20  and_then  \
  match  \
  n  sum  \
  Box::Error  description  cause  \
  \
```

```
    21  Ok  sum  main  \
```

```
    Err  err  \
  \
```

```
    \
return  18  13 17  \
  \
  unwrap  \
```

```
    trait  \
  9-22  \
```

```
    9-22  CliError
```

```

1. use std::io;
2. use std::num;
3. use std::fmt;
4. #[derive(Debug)]
5. enum CliError {
6.     Io(io::Error),
7.     Parse(num::ParseIntError),
8. }
9. impl fmt::Display for CliError {
10.     fn fmt(&self, f: &mut fmt::Formatter) -> fmt::Result {
11.         match *self {
12.             CliError::Io(ref err) => write!(f, "IO error: {}", err),
13.             CliError::Parse(ref err) => write!(f, "Parse error: {}"
14.                 , err),
15.         }
16.     }
17. }
18. impl Error for CliError {
19.     fn description(&self) -> &str {
20.         match *self {
21.             CliError::Io(ref err) => err.description(),
22.             CliError::Parse(ref err) => Error::description(err),
23.         }
24.     }
25.     fn cause(&self) -> Option<&Error> {
26.         match *self {
27.             CliError::Io(ref err) => Some(err),
28.             CliError::Parse(ref err) => Some(err),
29.         }
30.     }
31. }
32. type ParseResult<i32> = Result<i32, CliError>;

```

9-22 CliError Io
 io Error Parse num ParseIntError I/O

Box CliError Display Error

Box 32 type ParseResult i32 Box Error
CliError

main run Box Error
CliError 9-23

9-23 run CliError

```
1. fn run(filename: &str) -> ParseResult<i32> {
2.     File::open(filename).map_err(CliError::Io)
3.     .and_then(|mut f|{
4.         let mut contents = String::new();
5.         f.read_to_string(&mut contents)
6.         .map_err(CliError::Io).map(|_|contents)
7.     })
8.     .and_then(|contents|{
9.         let mut sum = 0;
10.        for c in contents.lines(){
11.            match c.parse::<i32>() {
12.                Ok(n) => {sum += n;},
13.                Err(err) => {
14.                    let err = CliError::Parse(err);
15.                    println!("Error Info: {} \n Cause by {:?}",
16.                        , err.description(), err.cause());
17.                },
18.                // Err(err) => {return Err(CliError::Parse(err));},
19.            }
20.        }
21.        Ok(sum)
22.    })
23. }
```

9-23 Box Error CliError
2 map_err |e|e.into map_err CliError Io
I/O

```
map_err(|_| CliError::Io(io::Error::new(CliError::
6
```

```
14 17 CliError::Parse(err)
return
18
```

```
Ok(sum)
main

```

Rust **try** Result 9-24
try

9-24 **try**

```
1. macro_rules! try {
2.     ($expr:expr) => (match $expr {
3.         $crate::result::Result::Ok(val) => val,
4.         $crate::result::Result::Err(err) => {
5.             return $crate::result::Result::Err(
6.                 $crate::convert::From::from(err)
7.             )
8.         }
9.     })
10. }
```

9-24 `macro_rules` "\$" 12 `match` Result return 6 `From::from`

`run` 9-25

9-25 **try** **run**


```

1. impl From<io::Error> for CliError {
2.     fn from(err: io::Error) -> CliError {
3.         CliError::Io(err)
4.     }
5. }
6. impl From<num::ParseIntError> for CliError {
7.     fn from(err: num::ParseIntError) -> CliError {
8.         CliError::Parse(err)
9.     }
10. }
11. fn run(filename: &str) -> ParseResult<i32> {
12.     let mut file = try!(File::open(filename));
13.     let mut contents = String::new();
14.     try!(file.read_to_string(&mut contents));
15.     let mut sum = 0;
16.     for c in contents.lines(){
17.         let n: i32 = try!(c.parse::<i32>());
18.         sum += n;
19.     }
20.     Ok(sum)
21. }

```

9-25 10 CliError From io Error CliError Io err num ParseIntError CliError Parse err try

12 try File open

14 try read_to_string UTF-8

17 try parse

try 17 try

try try try... Rust

9-26 try

9-26 try

```
1. fn run(filename: &str) -> ParseResult<i32> {
2.     let mut file = File::open(filename)?;
3.     let mut contents = String::new();
4.     file.read_to_string(&mut contents)?;
5.     let mut sum = 0;
6.     for c in contents.lines(){
7.         let n: i32 = c.parse::<i32>()?;
8.         sum += n;
9.     }
10.    Ok(sum)
11. }
```

9-26 try

Option T Result T E

run main

9-27 main

```
1. let args: Vec<String> = env::args().collect();
2. let filename = &args[1];
3. println!("In file {}", filename);
```

9-27 env args

env args nth nth Option T

9-28 nth main

```

1. fn main() {
2.     let filename = env::args().nth(1);
3.     match run(filename) {
4.         Ok(n) => {
5.             println!("{:?}", n);
6.         },
7.         Err(e) => {
8.             println!("main error: {}", e);
9.             process::exit(1);
10.        }
11.    }
12. }

```

9-28 `nth` 返回 `Option<String>` 类型，如果 `env::args()` 返回的 `String` 列表长度为 1，则返回 `None`，否则返回 `Some(filename)`。

9-29 `run` 函数

9-29 `run` 函数

```

1. use std::option::NoneError;
2. [derive(Debug)]
3. enum CliError {
4.     .....
5.     NoneError(NoneError),
6. }
7. impl fmt::Display for CliError {
8.     fn fmt(&self, f: &mut fmt::Formatter) -> fmt::Result {
9.         match *self {
10.             .....
11.             CliError::NoneError(ref err) =>
12.                 write!(f, "Command args error: {:?} ", err),
13.         }
14.     }
15. }
16. impl Error for CliError {
17.     fn description(&self) -> &str {
18.         match *self {
19.             ... ..
20.             CliError::NoneError(ref err) => "NoneError",
21.         }
22.     }
23.     fn cause(&self) -> Option<&Error> {
24.         match *self {
25.             .....
26.             _ => None,
27.         }
28.     }
29. }
30. impl From<NoneError> for CliError {
31.     fn from(err: NoneError) -> CliError {
32.         CliError::NoneError(err)
33.     }
34. }
35. fn run(filename: Option<String>) -> ParseResult<i32> {
36.     let mut file = File::open(filename)?;
37.     .....
38. }

```

9-29 使用 `Option` 类型

```
fn run(filename: Option<String>) -> Result<T, NoneError> {
    match filename {
        Some(f) => {
            // ...
        }
        None => {
            // ...
        }
    }
}
```

使用 `Option` 类型，从 `NoneError` 中返回 `CliError` 30-34

```
fn main() -> Result<T, NoneError> {
    match run(filename) {
        Ok(n) => {
            println!("{}", n);
        }
        Err(e) => {
            // ...
        }
    }
}
```

使用 `Option` 类型，从 `NoneError` 中返回 `CliError` 30-34

main 函数返回 Result

Rust 2018 使用 `main` 函数返回 `Result` 类型 9-29

9-30 main 函数返回 Result 类型

```
1. fn main() -> Result<(), i32> {
2.     let filename = env::args().nth(1);
3.     match run(filename) {
4.         Ok(n) => {
5.             println!("{}", n);
6.             return Ok(());
7.         },
8.         Err(e) => {
9.             return Err(1);
10.        }
11.    }
12. }
```

9-30 使用 `main` 函数返回 `Result` 类型 `i32` 类型，使用 `std::process::Termination` trait

main 返回 Result<T, E> 类型，E 为错误类型，bool 类型，never 类型 trait

9-30 编译选项

```
$ ./io_option test_txt
```

```
6
```

```
$ ./io_option test_txt1
```

```
Error: 1
```

编译选项 6 编译选项 1 编译选项

编译选项 main 返回 Result<T, E> 类型，bool 类型，never 类型

trait

trait std::ops::Try 9-30 编译选项

9-31 std::ops::Try

```
1. pub trait Try {
2.     type Ok;
3.     type Error;
4.     fn into_result(self) -> Result<Self::Ok, Self::Error>;
5.     fn from_error(v: Self::Error) -> Self;
6.     fn from_ok(v: Self::Ok) -> Self;
7. }
```

9-31 Try trait 编译选项 Ok 编译选项 Error 编译选项 into_result 编译选项 from_error 编译选项 from_ok 编译选项 Option<T> Try 编译选项 9-32

9-32 Option<T> std::ops::Try

Rust Panic Safety Rust
None unwrap 0
Safe Rust Rust UnwindSafe trait
Unsafe Rust Rust
13 Unsafe Rust

Rust catch_unwind Rust
catch_unwind
UnwindSafe catch_unwind Rust
catch_unwind catch_unwind
trait catch_unwind
abort

9-33 catch_unwind

9-33 catch_unwind

```
1. use std::panic;
2. fn sum(a: i32, b: i32) -> i32{
3.     a + b
4. }
5. fn main() {
6.     let result = panic::catch_unwind(|| { println!("hello!"); });
7.     assert!(result.is_ok());
8.     let result = panic::catch_unwind(|| { panic!("oh no!"); });
9.     assert!(result.is_err());
10.    println!("{}", sum(1, 2));
11. }
```

9-33 6 catch_unwind

8 catch_unwind panic
catch_unwind 9 10


```
thread 'main' panicked at 'oh no!', src/main.rs:11:8
```

```
note: Run with `RUST_BACKTRACE=1` for a backtrace.
```

```
----- standard output
```

```
hello!
```

```
3
```

```

std::panic::set_hook
9-34

```

```
9-34::set_hook
```

```
1. use std::panic;
2. fn sum(a: i32, b: i32) -> i32{
3.     a + b
4. }
5. fn main() {
6.     let result = panic::catch_unwind(|| { println!("hello!"); });
7.     assert!(result.is_ok());
8.     panic::set_hook(Box::new(|panic_info| {
9.         if let Some(location) = panic_info.location() {
10.             println!("panic occurred '{}' at {}",
11.                 location.file(), location.line()
12.             );
13.         } else {
14.             println!("can't get location information...");
15.         }
16.     }));
17.     let result = panic::catch_unwind(|| { panic!("oh no!"); });
18.     assert!(result.is_err());
19.     println!("{}", sum(1, 2));
20. }
```

```

9-34::set_hook
panic_info::location

```

```
hello!
```

```
panic occurred 'src/main.rs' at 18
```

```
3
```

set_hook 钩子函数，用于注册钩子函数。
take_hook 钩子函数，用于取消钩子函数。

9.5 钩子

Rust 钩子函数，用于注册钩子函数。Error 钩子函数，用于注册钩子函数。
Rust 钩子函数，用于注册钩子函数。crate 钩子函数，用于注册钩子函数。
error-chain failure 钩子函数，用于注册钩子函数。

failure 钩子函数，用于注册钩子函数。
cargo new 钩子函数，用于注册钩子函数。

```
$ cargo new failure_crate
```

Rust 钩子函数，用于注册钩子函数。Cargo 钩子函数，用于注册钩子函数。
Bin 钩子函数，用于注册钩子函数。

failure_crate 钩子函数，用于注册钩子函数。cargo.toml 钩子函数，用于注册钩子函数。
9-35 钩子函数。

9-35 cargo.toml

1. [dependencies]
2. failure="0.1.2"
3. failure derive="0.1.2"

cargo.toml 钩子函数，用于注册钩子函数。failure 钩子函数，用于注册钩子函数。
failure_derive 钩子函数，用于注册钩子函数。

src/main.rs 钩子函数，用于注册钩子函数。9-36 钩子函数。

9-36 src/main.rs

1. extern crate failure;
2. #[macro_use] extern crate failure_derive;
3. use failure::{Context, Fail, Backtrace};
4. use std::env;
5. use std::fs::File;
6. use std::io::prelude::*;

9-36 钩子函数，用于注册钩子函数。failure 钩子函数，用于注册钩子函数。
failure Context Fail Backtrace 钩子函数，用于注册钩子函数。


```

1. impl From<std::io::Error> for Error {
2.     fn from(err: std::io::Error) -> Error {
3.         Error {
4.             inner: Context::new(
5.                 ErrorKind::Io(err, Backtrace::default())
6.             )
7.         }
8.     }
9. }
10. impl From<std::num::ParseIntError> for Error {
11.     fn from(err: std::num::ParseIntError) -> Error {
12.         Error { inner: Context::new(ErrorKind::Parse(err)) }
13.     }
14. }
15. type ParseResult<i32> = Result<i32, Error>;

```

9-39 From Error std::io::Error std::num::ParseIntError

type ParseResult<i32> Result<i32, Error> Fail trait failure_crate

failure failure

9.6

Rust

Rust “” Rust “” Fast Fail Bug assert debug_assert assert

```
Rust
Option<T> Result<T> E
" " Result<T> E
Rust
Result<T> E Rust catch_unwind
```

```
Rust Result<T> E
```

```
Rust
error_chain failure error_chain
failure Error Rust
failure
```

```
Rust
Rust
```

第10章 包管理

包管理是软件开发中不可或缺的一部分。

本章将介绍包管理的基本概念，并重点介绍 GitHub 包管理。GitHub 包管理是一种基于 GitHub 的包管理方式，它允许开发者在 GitHub 上发布和管理他们的包。GitHub 包管理的主要优点是简单易用，且与 GitHub 生态系统无缝集成。

本章将介绍包管理的基本概念，并重点介绍 GitHub 包管理。GitHub 包管理是一种基于 GitHub 的包管理方式，它允许开发者在 GitHub 上发布和管理他们的包。GitHub 包管理的主要优点是简单易用，且与 GitHub 生态系统无缝集成。

- 包管理是软件开发中不可或缺的一部分。
- 本章将介绍包管理的基本概念，并重点介绍 GitHub 包管理。
- GitHub 包管理是一种基于 GitHub 的包管理方式。

本章将介绍包管理的基本概念，并重点介绍 GitHub 包管理。GitHub 包管理是一种基于 GitHub 的包管理方式，它允许开发者在 GitHub 上发布和管理他们的包。GitHub 包管理的主要优点是简单易用，且与 GitHub 生态系统无缝集成。

本章将介绍包管理的基本概念，并重点介绍 GitHub 包管理。GitHub 包管理是一种基于 GitHub 的包管理方式，它允许开发者在 GitHub 上发布和管理他们的包。GitHub 包管理的主要优点是简单易用，且与 GitHub 生态系统无缝集成。

Rust 包管理工具 Cargo 是 Rust 语言的标准包管理工具。Cargo 允许开发者在 Rust 项目中发布和管理包。Cargo 包管理工具与 GitHub 包管理工具 crates.io 无缝集成。

10.1 简介

crate 是 Rust 的包管理工具，它负责管理 crate 的依赖关系。Cargo 是 Rust 的包管理工具，它负责管理 crate 的依赖关系。

- 管理 crate 的 metadata
- 管理 crate 的依赖关系
- 管理 crate 的 rustc 编译选项
- 管理 Rust 的编译选项
- 管理 Cargo 的编译选项

10.1.1 Cargo 简介

使用 cargo new 创建一个新的 crate

```
$ cargo new csv-read --lib
Created library `csv-read` project
```

使用 tree 命令查看 crate 的结构

```
$ tree csv-read
csv-read
├── Cargo.toml
└── src
    └── lib.rs
```

Cargo.toml 和 src/lib.rs 是 crate 的两个主要文件。Cargo.toml 是 crate 的配置文件，它定义了 crate 的名称、版本、依赖关系等信息。TOML 是一种轻量级的配置文件格式，它比 JSON 更简单，比 XML 更灵活。[\[1\]](#) TOML 的官方网站是 <https://toml.io/>。Cargo.toml 的官方网站是 <https://doc.rust-lang.org/cargo/commands/cargo-new.html>。

10-1 Cargo.toml 文件


```

1. [package]
2.   name = "csv-read"
3.   version = "0.1.0"
4.   authors = ["Your Name <you@email.com>"]
5.   edition = "2018"
6. [dependencies]

```

10-1 Cargo.toml manifest file
 15 "csv-read"

Rust 1.30 crate **edition** "2018"
 "2018" crate **Rust 2018**
 "2015" **Rust 2015**

src/lib.rs 10-2
 10-2 src/lib.rs

```

1. #[cfg(test)]
2. mod tests {
3.     #[test]
4.     fn it_works() {
5.         assert_eq!(2 + 2, 4);
6.     }
7. }

```

src/lib.rs tests Rust mod
 [cfg(test)] cargo test
 tests it_works cargo test

```
Compiling csv-read v0.1.0 (file:///LocalPath/to/csv-read)
  Finished dev [unoptimized + debuginfo] target(s) in 0.78 secs
  Running target/debug/deps/csv_read-1716337329e24fc6
running 1 test
test tests::it_works ... ok
test result:
    ok. 1 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out
Doc-tests csv-read
running 0 tests
test result:
    ok. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out
```

```

testsit_works"ok.1 passed"
Doc-testsRust

```

```
tree
```

```

.
├── Cargo.lock
├── Cargo.toml
├── src
│   └── lib.rs
└── target
    └── ...
```

```

Cargo.lock targetCargo.lock
Cargo.toml
```

```

· Cargo.toml
· Cargo.lockCargo
```

```

targetDebug
--release
```

```

cargo new
--bin
```

```
$ cargo new --bin csv-read
Created binary (application) `csv-read` project
```

--bin 指定可执行文件的名称。tree 命令显示目录结构。

```
$ tree csv-read
csv-read
├── Cargo.toml
└── src
    └── main.rs
```

src/main.rs 10-3 main.rs

10-3 src/main.rs

```
1. fn main() {
2.     println!("Hello, world!");
3. }
```

main.rs 编译 main 函数。cargo build 编译并生成可执行文件。cargo run 运行可执行文件。

```
$ cargo run
Compiling csv-read v0.1.0 (file:///LocalPath/to/csv-replace)
Finished dev [unoptimized + debuginfo] target(s) in 1.35 secs
Running `target/debug/csv-read`
Hello, world!
```

10.1.2 依赖项

Rust 项目使用 Cargo.toml 文件中的 **[dependencies]** 部分来声明依赖项。

csv-read 依赖 linked-list 10-4

10-4 Cargo.toml 中的 **linked-list**

```
1. [dependencies]
2. linked-list = "0.0.3"
```

src/main.rs src/lib.rs extern crate
10-5

10-5 src/main.rs extern crate

```
1. extern crate linked_list;
2. fn main() {
3.     println!("Hello, world!");
4. }
```

10-5 1 **extern crate** Rust
2015 **Rust 2018** extern crate
Cargo.toml

extern crate linked_list
"_" Cargo.toml "-" Cargo
linked-list linked_list

Rust "-rs" "_rs"
"

regex

Rust regex
cargo new--bin use_regex Cargo.toml
regex 10-6

10-6 Cargo.toml regex

```
1. [dependencies]
2. regex = "1.0.5"
```

regex 1.0.5 src/main.rs extern crate
regex 10-7

10-7 src/main.rs regex


```

1. fn main() {
2.     let re = Regex::new(r"(?x)
3.         (?P<year>\d{4}) # the year
4.         -
5.         (?P<month>\d{2}) # the month
6.         -
7.         (?P<day>\d{2}) # the day
8.     ").unwrap();
9.     let caps = re.captures("2018-01-01").unwrap();
10.    assert_eq!("2018", &caps["year"]);
11.    assert_eq!("01", &caps["month"]);
12.    assert_eq!("01", &caps["day"]);
13.    let after = re.replace_all("2018-01-01", "$month/$day/$year");
14.    assert_eq!(after, "01/01/2018");
15. }

```

10-8 2 8 Regex new x
 x regex

- **i**
- **m** “^” “\$”
- **s** “.” “\n”
- **U** “x*” “x*”
- **u** Unicode
- **x** “”

10-8 2 8 P name exp

9 captures HashMap
 HashMap 10 12 caps

13 14 replace_all
 “\$”

regex

crate lazy_static

crate Rust crate

Constant Static crate

• crate HashMap Vector

• crate

• crate

• crate UnsafeCell crate

• crate Sync crate

• crate

crate HashMap Vector lazy_static

lazy Cargo.toml lazy_static 10-9

10-9 Cargo.toml lazy_static

1. [dependencies]
2. regex = "1.0.5"
3. lazy_static = "1.1.0"

use_regex lazy_static src/main.rs extern crate lazy_static 10-10

10-10 src/main.rs extern crate lazy_static

```

1. #[macro_use] extern crate lazy_static;
2. extern crate regex;
3. use regex::Regex;
4. lazy_static! {
5.     static ref RE: Regex = Regex::new(r"(?x)
6.         (?P<year>\d{4})- # the year
7.         (?P<month>\d{2})- # the month
8.         (?P<day>\d{2}) # the day
9.     ").unwrap();
10.    static ref EMAIL_RE: Regex = Regex::new(r"(?x)
11.        ^\w+@(?:gmail|163|qq)\.(:com|cn|com\.cn|net)$
12.    ").unwrap();
13. }
14. fn regex_date(text: &str) -> regex::Captures {
15.    RE.captures(text).unwrap()
16. }
17. fn regex_email(text: &str) -> bool {
18.    EMAIL_RE.is_match(text)
19. }
20. fn main() {
21.    let caps = regex_date("2018-01-01");
22.    assert_eq!("2018", &caps["year"]);
23.    assert_eq!("01", &caps["month"]);
24.    assert_eq!("01", &caps["day"]);
25.    let after = RE.replace_all("2018-01-01", "$month/$day/$year");
26.    assert_eq!(after, "01/01/2018");
27.    assert!(regex_email("alex@gmail.com"), true);
28.    assert_eq!(regex_email("alex@gmail.cn.com"), false);
29. }

```

10-10 Rust 2015 1 **[macro_use]**
extern crate lazy_static lazy_static
 lazy_static[macro_use]extern crate lazy_static
 [macro_use] Rust 2018

extern crate `lazy_static` **[macro_use]** `RE` `EMAIL_RE`
`2`

`4` `13` `lazy_static` `RE` `EMAIL_RE`
`1` `2` `3` `4` `5` `6` `7` `8` `9` `10` `11` `12` `13` `14` `15` `16` `17` `18` `19` `20` `21` `22` `23` `24` `25` `26` `27` `28` `29` `30` `31` `32` `33` `34` `35` `36` `37` `38` `39` `40` `41` `42` `43` `44` `45` `46` `47` `48` `49` `50` `51` `52` `53` `54` `55` `56` `57` `58` `59` `60` `61` `62` `63` `64` `65` `66` `67` `68` `69` `70` `71` `72` `73` `74` `75` `76` `77` `78` `79` `80` `81` `82` `83` `84` `85` `86` `87` `88` `89` `90` `91` `92` `93` `94` `95` `96` `97` `98` `99` `100` `101` `102` `103` `104` `105` `106` `107` `108` `109` `110` `111` `112` `113` `114` `115` `116` `117` `118` `119` `120` `121` `122` `123` `124` `125` `126` `127` `128` `129` `130` `131` `132` `133` `134` `135` `136` `137` `138` `139` `140` `141` `142` `143` `144` `145` `146` `147` `148` `149` `150` `151` `152` `153` `154` `155` `156` `157` `158` `159` `160` `161` `162` `163` `164` `165` `166` `167` `168` `169` `170` `171` `172` `173` `174` `175` `176` `177` `178` `179` `180` `181` `182` `183` `184` `185` `186` `187` `188` `189` `190` `191` `192` `193` `194` `195` `196` `197` `198` `199` `200` `201` `202` `203` `204` `205` `206` `207` `208` `209` `210` `211` `212` `213` `214` `215` `216` `217` `218` `219` `220` `221` `222` `223` `224` `225` `226` `227` `228` `229` `230` `231` `232` `233` `234` `235` `236` `237` `238` `239` `240` `241` `242` `243` `244` `245` `246` `247` `248` `249` `250` `251` `252` `253` `254` `255` `256` `257` `258` `259` `260` `261` `262` `263` `264` `265` `266` `267` `268` `269` `270` `271` `272` `273` `274` `275` `276` `277` `278` `279` `280` `281` `282` `283` `284` `285` `286` `287` `288` `289` `290` `291` `292` `293` `294` `295` `296` `297` `298` `299` `300` `301` `302` `303` `304` `305` `306` `307` `308` `309` `310` `311` `312` `313` `314` `315` `316` `317` `318` `319` `320` `321` `322` `323` `324` `325` `326` `327` `328` `329` `330` `331` `332` `333` `334` `335` `336` `337` `338` `339` `340` `341` `342` `343` `344` `345` `346` `347` `348` `349` `350` `351` `352` `353` `354` `355` `356` `357` `358` `359` `360` `361` `362` `363` `364` `365` `366` `367` `368` `369` `370` `371` `372` `373` `374` `375` `376` `377` `378` `379` `380` `381` `382` `383` `384` `385` `386` `387` `388` `389` `390` `391` `392` `393` `394` `395` `396` `397` `398` `399` `400` `401` `402` `403` `404` `405` `406` `407` `408` `409` `410` `411` `412` `413` `414` `415` `416` `417` `418` `419` `420` `421` `422` `423` `424` `425` `426` `427` `428` `429` `430` `431` `432` `433` `434` `435` `436` `437` `438` `439` `440` `441` `442` `443` `444` `445` `446` `447` `448` `449` `450` `451` `452` `453` `454` `455` `456` `457` `458` `459` `460` `461` `462` `463` `464` `465` `466` `467` `468` `469` `470` `471` `472` `473` `474` `475` `476` `477` `478` `479` `480` `481` `482` `483` `484` `485` `486` `487` `488` `489` `490` `491` `492` `493` `494` `495` `496` `497` `498` `499` `500` `501` `502` `503` `504` `505` `506` `507` `508` `509` `510` `511` `512` `513` `514` `515` `516` `517` `518` `519` `520` `521` `522` `523` `524` `525` `526` `527` `528` `529` `530` `531` `532` `533` `534` `535` `536` `537` `538` `539` `540` `541` `542` `543` `544` `545` `546` `547` `548` `549` `550` `551` `552` `553` `554` `555` `556` `557` `558` `559` `560` `561` `562` `563` `564` `565` `566` `567` `568` `569` `570` `571` `572` `573` `574` `575` `576` `577` `578` `579` `580` `581` `582` `583` `584` `585` `586` `587` `588` `589` `590` `591` `592` `593` `594` `595` `596` `597` `598` `599` `600` `601` `602` `603` `604` `605` `606` `607` `608` `609` `610` `611` `612` `613` `614` `615` `616` `617` `618` `619` `620` `621` `622` `623` `624` `625` `626` `627` `628` `629` `630` `631` `632` `633` `634` `635` `636` `637` `638` `639` `640` `641` `642` `643` `644` `645` `646` `647` `648` `649` `650` `651` `652` `653` `654` `655` `656` `657` `658` `659` `660` `661` `662` `663` `664` `665` `666` `667` `668` `669` `670` `671` `672` `673` `674` `675` `676` `677` `678` `679` `680` `681` `682` `683` `684` `685` `686` `687` `688` `689` `690` `691` `692` `693` `694` `695` `696` `697` `698` `699` `700` `701` `702` `703` `704` `705` `706` `707` `708` `709` `710` `711` `712` `713` `714` `715` `716` `717` `718` `719` `720` `721` `722` `723` `724` `725` `726` `727` `728` `729` `730` `731` `732` `733` `734` `735` `736` `737` `738` `739` `740` `741` `742` `743` `744` `745` `746` `747` `748` `749` `750` `751` `752` `753` `754` `755` `756` `757` `758` `759` `760` `761` `762` `763` `764` `765` `766` `767` `768` `769` `770` `771` `772` `773` `774` `775` `776` `777` `778` `779` `780` `781` `782` `783` `784` `785` `786` `787` `788` `789` `790` `791` `792` `793` `794` `795` `796` `797` `798` `799` `800` `801` `802` `803` `804` `805` `806` `807` `808` `809` `810` `811` `812` `813` `814` `815` `816` `817` `818` `819` `820` `821` `822` `823` `824` `825` `826` `827` `828` `829` `830` `831` `832` `833` `834` `835` `836` `837` `838` `839` `840` `841` `842` `843` `844` `845` `846` `847` `848` `849` `850` `851` `852` `853` `854` `855` `856` `857` `858` `859` `860` `861` `862` `863` `864` `865` `866` `867` `868` `869` `870` `871` `872` `873` `874` `875` `876` `877` `878` `879` `880` `881` `882` `883` `884` `885` `886` `887` `888` `889` `890` `891` `892` `893` `894` `895` `896` `897` `898` `899` `900` `901` `902` `903` `904` `905` `906` `907` `908` `909` `910` `911` `912` `913` `914` `915` `916` `917` `918` `919` `920` `921` `922` `923` `924` `925` `926` `927` `928` `929` `930` `931` `932` `933` `934` `935` `936` `937` `938` `939` `940` `941` `942` `943` `944` `945` `946` `947` `948` `949` `950` `951` `952` `953` `954` `955` `956` `957` `958` `959` `960` `961` `962` `963` `964` `965` `966` `967` `968` `969` `970` `971` `972` `973` `974` `975` `976` `977` `978` `979` `980` `981` `982` `983` `984` `985` `986` `987` `988` `989` `990` `991` `992` `993` `994` `995` `996` `997` `998` `999` `1000` `1001` `1002` `1003` `1004` `1005` `1006` `1007` `1008` `1009` `1010` `1011` `1012` `1013` `1014` `1015` `1016` `1017` `1018` `1019` `1020` `1021` `1022` `1023` `1024` `1025` `1026` `1027` `1028` `1029` `1030` `1031` `1032` `1033` `1034` `1035` `1036` `1037` `1038` `1039` `1040` `1041` `1042` `1043` `1044` `1045` `1046` `1047` `1048` `1049` `1050` `1051` `1052` `1053` `1054` `1055` `1056` `1057` `1058` `1059` `1060` `1061` `1062` `1063` `1064` `1065` `1066` `1067` `1068` `1069` `1070` `1071` `1072` `1073` `1074` `1075` `1076` `1077` `1078` `1079` `1080` `1081` `1082` `1083` `1084` `1085` `1086` `1087` `1088` `1089` `1090` `1091` `1092` `1093` `1094` `1095` `1096` `1097` `1098` `1099` `1100` `1101` `1102` `1103` `1104` `1105` `1106` `1107` `1108` `1109` `1110` `1111` `1112` `1113` `1114` `1115` `1116` `1117` `1118` `1119` `1120` `1121` `1122` `1123` `1124` `1125` `1126` `1127` `1128` `1129` `1130` `1131` `1132` `1133` `1134` `1135` `1136` `1137` `1138` `1139` `1140` `1141` `1142` `1143` `1144` `1145` `1146` `1147` `1148` `1149` `1150` `1151` `1152` `1153` `1154` `1155` `1156` `1157` `1158` `1159` `1160` `1161` `1162` `1163` `1164` `1165` `1166` `1167` `1168` `1169` `1170` `1171` `1172` `1173` `1174` `1175` `1176` `1177` `1178` `1179` `1180` `1181` `1182` `1183` `1184` `1185` `1186` `1187` `1188` `1189` `1190` `1191` `1192` `1193` `1194` `1195` `1196` `1197` `1198` `1199` `1200` `1201` `1202` `1203` `1204` `1205` `1206` `1207` `1208` `1209` `1210` `1211` `1212` `1213` `1214` `1215` `1216` `1217` `1218` `1219` `1220` `1221` `1222` `1223` `1224` `1225` `1226` `1227` `1228` `1229` `1230` `1231` `1232` `1233` `1234` `1235` `1236` `1237` `1238` `1239` `1240` `1241` `1242` `1243` `1244` `1245` `1246` `1247` `1248` `1249` `1250` `1251` `1252` `1253` `1254` `1255` `1256` `1257` `1258` `1259` `1260` `1261` `1262` `1263` `1264` `1265` `1266` `1267` `1268` `1269` `1270` `1271` `1272` `1273` `1274` `1275` `1276` `1277` `1278` `1279` `1280` `1281` `1282` `1283` `1284` `1285` `1286` `1287` `1288` `1289` `1290` `1291` `1292` `1293` `1294` `1295` `1296` `1297` `1298` `1299` `1300` `1301` `1302` `1303` `1304` `1305` `1306` `1307` `1308` `1309` `1310` `1311` `1312` `1313` `1314` `1315` `1316` `1317` `1318` `1319` `1320` `1321` `1322` `1323` `1324` `1325` `1326` `1327` `1328` `1329` `1330` `1331` `1332` `1333` `1334` `1335`

```
1. #[macro_use]extern crate lazy_static;
2. mod static_kv {
3.     use std::collections::HashMap;
4.     use std::sync::RwLock;
5.     pub const NF: &'static str = "not found";
6.     lazy_static! {
7.         pub static ref MAP: HashMap<u32, &'static str> = {
8.             let mut m = HashMap::new();
9.             m.insert(0, "foo");
10.            m
11.        };
12.        pub static ref MAP_MUT: RwLock<HashMap<u32, &'static str>> =
13.            {
14.                let mut m = HashMap::new();
15.                m.insert(0, "bar");
16.                RwLock::new(m)
17.            };
18.    }
19. }
20. fn read_kv() {
21.    let ref m = static_kv::MAP;
22.    assert_eq!("foo", *m.get(&0).unwrap_or(&static_kv::NF));
23.    assert_eq!(static_kv::NF,
24.        *m.get(&1).unwrap_or(&static_kv::NF));
25. }
26. fn rw_mut_kv() -> Result<(), String> {
27.    {
```

```

28.         let m = static_kv::MAP_MUT
29.             .read().map_err(|e| e.to_string())?;
30.         assert_eq!("bar", *m.get(&0).unwrap_or(&static_kv::NF));
31.     }
32.     {
33.         let mut m = static_kv::MAP_MUT
34.             .write().map_err(|e| e.to_string())?;
35.         m.insert(1, "baz");
36.     }
37.     Ok(())
38. }
39. fn main() {
40.     read_kv();
41.     match rw_mut_kv() {
42.         Ok(()) => {
43.             let m = static_kv::MAP_MUT
44.                 .read().map_err(|e| e.to_string()).unwrap();
45.             assert_eq!("baz", *m.get(&1).unwrap_or(&static_kv::NF));
46.         },
47.         Err(e) => {println!("Error {}", e)},
48.     }
49. }

```

10-11 1 [macro_use] extern crate lazy_static lazy_static

2 19 mod static_kv Rust static_kv pub

3 4 use std collections std sync HashMap RwLock static_kv

5 pub const NF static_hash NF

6 18 lazy_static MAP MAP_MUT HashMap HashMap lazy_static

10-12

10-12 lazy_static

```
1. lazy_static! {  
2.     [pub] static ref NAME_1: TYPE_1 = EXPR_1;  
3.     [pub] static ref NAME_2: TYPE_2 = EXPR_2;  
4.     ...  
5.     [pub] static ref NAME_N: TYPE_N = EXPR_N;  
6. }
```

lazy_static

10-11 7 11 HashMap
MAP "{0foo}"

12 17 RwLock HashMap u32 &
static str MAP_MUT RwLock
HashMap HashMap HashMap
Sync HashMap HashMap HashMap
Mutex HashMap

· RwLock

· Mutex

RwLock
11

MAP_MUT RwLock

20 25 read_kv static_kv MAP
static_kv MAP static_kv
MAP

21 ref static_kv MAP m
&static_kv MAP m 22 24 m
HashMap get MAP get
HashMap get Option T
None &static_kv NF

crate 的 version 是 "0.0.0" 那么 crate 的 version 是 "0.0.0"

crate lazy_static 的 version 是 "1.0.0" 那么 crate lazy_static 的 version 是 "1.0.0" Cargo 的 version 是 "1.1.0" 那么 cargo build cargo run 的 version 是 "1.1.0"

crate 的 version 是 "0.0.0"

· crate ^ 的 version 是 [major minor patch] 的 version 是 [major minor patch]

· crate * 的 version 是 [major minor patch] 的 version 是 [major minor patch]

· crate 的 version 是 [major minor patch] 的 version 是 [major minor patch]

· crate 的 version 是 [major minor patch] 的 version 是 [major minor patch]

crate 的 version 是 10-13

crate 10-13 的 version 是 10-13

1. // := 表示 等价于
2. // 补注号示例
3. ^1.2.3 := >=1.2.3 <2.0.0
4. ^1.2 := >=1.2.0 <2.0.0
5. ^1 := >=1.0.0 <2.0.0
6. ^0.2.3 := >=0.2.3 <0.3.0
7. ^0.0.3 := >=0.0.3 <0.0.4
8. ^0.0 := >=0.0.0 <0.1.0
9. ^0 := >=0.0.0 <1.0.0
10. // 通配符示例
11. := >=0.0.0
12. 1.* := >=1.0.0 <2.0.0
13. 1.2.* := >=1.2.0 <1.3.0

```

14. // 波浪线示例
15. ~1.2.3 := >=1.2.3 <1.3.0
16. ~1.2 := >=1.2.0 <1.3.0
17. ~1 := >=1.0.0 <2.0.0
18. // 手动指定
19. >= 1.2.0
20. > 1
21. < 2
22. = 1.2.3
23. // 手动指定多个版本
24. >= 1.2, < 1.5.

```

使用 Cargo 安装 git 包并运行 10-14

10-14 使用 git 包

1. [dependencies]
2. rand = { git = "https://github.com/rust-lang-nursery/rand" }

使用 static_hashmap 包并运行 10-15

10-15 使用 path 包和 hello_world

1. [dependencies]
2. hello_world = { path = "hello_world", version = "0.1.0" }

使用 10-15 中的 hello_world 包和 static_hashmap 包并运行 path 包并运行 static_hashmap 包并运行 path 包并运行 crates.io

10.1.3 Cargo.toml

TOML 包并运行 Cargo TOML 包并运行 regex [\[3\]](#) 包并运行 10-16 包并运行 regex

10-16 使用 regex 包

```

regex
├── bench/
├── ci/
├── examples/
├── regex-capi/
├── regex-debug/
├── regex-syntax/
├── scripts/
├── src/
├── tests/
└── Cargo.toml

```

`regex` `bench` `regex-capi` `regex-debug`
`regex-syntax`

[package]

`Cargo.toml` 10-17 `regex`
`Cargo.toml` `[package]`

10-17 `regex` `Cargo.toml` `[package]`

1. `[package]`
2. `name = "regex"`
3. `version = "1.0.5" #:version`
4. `authors = ["The Rust Project Developers"]`
5. `license = "MIT/Apache-2.0"`
6. `readme = "README.md"`
7. `repository = "https://github.com/rust-lang/regex"`
8. `documentation = "https://docs.rs/regex"`
9. `homepage = "https://github.com/rust-lang/regex"`
10. `description = ""`
11. An implementation of regular expressions for Rust.
12. This implementation uses
13. finite automata and guarantees linear time matching on all inputs.
14. ""
15. `categories = ["text-processing"]`

TOML 的 `[package]` 表 **Table** 的 `[package]` 表
包含 `regex` 表中的 `name`、`authors`、`repository`、`documentation`、`description`、`categories`
“

`[package]` 表中的 `version` 为 10-18 的 JSON
10-18 `[package]` 的 JSON

```
1. "package": {
2.     "name": "regex",
3.     "version": "1.0.5",
4.     // 省略
5.     "categories": ["text-processing"]
6. }
```

[badges]

`regex` 的 `Cargo.toml` 中的 **[badges]** 表 10-19

10-19 **[badges]**

```
1. [badges]
2.   travis-ci = { repository = "rust-lang/regex" }
3.   appveyor = { repository = "rust-lang-libs/regex" }
```

10-19 的 **[badges]** 表中的 `travis-ci`、`appveyor`、`crates.io`、`travis-ci`、`appveyor`、`travis-ci`、`appveyor`、`Linux`、`Mac OS`、`Windows`、`[badges]`、`GitLab`、`codecov`、`[badges]`

[workspace]

`[workspace]` 表 10-20

10-20 **[workspace]**

```
1. [workspace]
2.   members = ["bench", "regex-capi", "regex-debug", "regex-syntax"]
```

10-20 `[workspace]` `Workspace`
`crate` `crate` `regex`
`members` `bench` `regex-capi` `regex-debug` `regex-syntax`

`Cargo.toml` `regex`
`Cargo.toml` `target` `Cargo.lock`

[dependencies]

`regex` `Cargo.toml` `[dependencies]`
10-21

10-21 [dependencies] [dev-dependencies]

1. `[dependencies]`
2. `aho-corasick = "0.6.7"`
3. `memchr = "2.0.2"`
4. `thread_local = "0.3.6"`
5. `regex-syntax = { path = "regex-syntax", version = "0.6.2" }`
6. `utf8-ranges = "1.0.1"`
7. `[dev-dependencies]`
8. `lazy_static = "1"`
9. `quickcheck = { version = "0.7", default-features = false }`
10. `rand = "0.5"`

10-21 **[dependencies]** **[dev-dependencies]** `[dependencies]`
`cargo build` `[dev-dependencies]`
`tests` `examples` `benchmarks`
cargo test **cargo bench**

[features]

[features] 10-22

10-22 [features]

```

1. [features]
2.   default = ["use_std"]
3.   use_std = []
4.   unstable = ["pattern"]
5.   pattern = []

```

10-22 **[features]** Rust **[cfg]** `std::Pattern` trait trait `unstable`

10-23 `regex`

10-23 `regex` **[cfg]**

```

1. #[cfg(not(feature = "use_std"))]
2.   compile_error!("`use_std` feature is currently required to build this crate");
3. #[cfg(feature = "pattern")]
4.   mod pattern;

```

10-23 3 **[cfg feature=pattern]** `cargo build--features pattern` Cargo Rust `rustc --cfg feature=pattern` `pattern`

1 **[cfg not feature=use_std]** **[cfg feature=use_std]** `features`

[lib]

`regex` Cargo.toml **[lib]** 10-24

10-24 **[lib]**

```

1. [lib]
2.   bench = false

```

10-24 **[lib]**

· **name** `name=foo` `libfoo.a` `libfoo.so`

· **crate-type** crate-type=[dylibstaticlib] 编译选项
编译选项

· **path** path=src/lib.rs 编译选项
src/lib.rs

· **test** test=true 编译选项

· **bench** bench=true 编译选项

编译选项bench
false

[test]

[[test]] 编译选项TOML 编译选项
10-25

10-25[[test]]

1. [[test]]
2. path = "tests/test_default.rs"
3. name = "default"
4. [[test]]
5. path = "tests/test_default_bytes.rs"
6. name = "default-bytes"
7. [[test]]
8. path = "tests/test_nfa.rs"
9. name = "nfa"

10-25[[test]]regexCargo.toml
[[test]]JSON 10-26

10-26[[test]]JSON

1. {
2. "test": [
3. { "path": "...", "name": "..." },
4. { "path": "...", "name": "..." },
5. { "path": "...", "name": "..." },
6.]
7. }

[[test]]
[lib]

[profile]

[profile] 10-27

10-27 [profile]

1. [profile.release]
2. debug = true
3. [profile.bench]
4. debug = true
5. [profile.test]
6. debug = true

Cargo rustc [profile]
profile

10-27 [profile].
[profile.release][profile.bench][profile.test]**JSON**
10-28

10-28 [profile] JSON

1. "profile"{
2. "release":{ "debug": "true"},
3. "bench":{ "debug": "true"},
4. "test":{ "debug": "true"},
5. }

Release Bench Test Cargo
[profile.dev] Debug Release Bench
Test Debug debug opt-
level to

Cargo.toml
bench Cargo.toml

bench 10-29

10-29 bench

```

bench
├─ log/
├─ src/
├─ Cargo.toml
├─ build.rs
├─ compile
└─ run

```

Bench 是 Rust 的基准测试工具，它使用 Cargo 来管理依赖。log 是 Rust 的日志库，src 是 Rust 的源代码目录。Cargo.toml 是 Cargo 的配置文件，Cargo 是 Rust 的包管理器。build.rs 是 Rust 的构建脚本，Build Script 是 Rust 的构建脚本。cargo build 是 Rust 的构建命令，Rust 是 Rust 的编译器。C 是 Rust 的编译器，Rust 是 Rust 的编译器。build.rs 是 Rust 的构建脚本，12 是 Rust 的编译器。compile 是 Rust 的编译器，run 是 Rust 的编译器。shell 是 Rust 的编译器。cargo build 是 Rust 的构建命令，cargo bench 是 Rust 的基准测试命令。

bench 是 Cargo.toml 中的一个配置项，它用于指定基准测试的脚本。10-30 是 Cargo.toml 中的一个配置项，它用于指定基准测试的脚本。

10-30 是 Cargo.toml 中的一个配置项，它用于指定基准测试的脚本。

1. [package]
2. ...
3. build = "build.rs"
4. workspace = ".."
5. [[bin]]
6. name = "regex-run-one"
7. path = "src/main.rs"
8. bench = false
9. [[bench]]
10. name = "bench"
11. path = "src/bench.rs"
12. test = false
13. bench = true

10-30 是 Cargo.toml 中的一个配置项，它用于指定基准测试的脚本。regex 是 Cargo.toml 中的一个配置项，它用于指定基准测试的脚本。

1 4 是 Cargo.toml 中的一个配置项，它用于指定基准测试的脚本。build 是 Cargo.toml 中的一个配置项，它用于指定基准测试的脚本。workspace 是 Cargo.toml 中的一个配置项，它用于指定基准测试的脚本。build.rs 是 Cargo.toml 中的一个配置项，它用于指定基准测试的脚本。bench 是 Cargo.toml 中的一个配置项，它用于指定基准测试的脚本。build.rs 是 Cargo.toml 中的一个配置项，它用于指定基准测试的脚本。workspace 是 Cargo.toml 中的一个配置项，它用于指定基准测试的脚本。regex 是 Cargo.toml 中的一个配置项，它用于指定基准测试的脚本。

Cargo.toml [workspace] => workspace “..” workspace

```
[[bin]]
path = "src/bin/main.rs"
name = "main"
[[bench]]
path = "src/bin/bench.rs"
name = "bench"
false
```

bench Cargo.toml crates.io

10.1.4 Cargo

Cargo \$HOME/.cargo/config Linux/ UNIX Windows %USERPROFILE%\cargo\config 10-31

10-31 \$HOME/.cargo/config

- [registry]
- token = "your_crates_io_token"
- [source.crates-io]
- registry = "https://github.com/rust-lang/crates.io-index"
- [alias]
- b = "build"
- t = "test"
- r = "run"
- rr = "run --release"
- ben = "bench"
- space_example = ["run", "--release", "--", "\"command list\""]

10-30 Cargo TOML [registry] crates.io token crates.io crate

运行 cargo fmt 命令，rustfmt 命令
rustfmt_skip 选项

rustfmt Cargo.toml 10-32

10-32 rustfmt Cargo.toml

1. [[bin]]
2. name = "rustfmt"
3. [[bin]]
4. name = "cargo-fmt"
5. [[bin]]
6. name = "rustfmt-format-diff"
7. [[bin]]
8. name = "git-rustfmt"

10-32 [[bin]] cargo-fmt cargo install rustfmt cargo fmt
rustfmt src/bin Rust rustfmt.rs cargo-fmt.rs rustfmt-format-diff.rs git-
rustfmt.rs [[bin]] path

rustfmt rustfmt.toml
10-33

10-33 rustfmt.toml

1. # 最大宽度
2. max_width = 90
3. # fn 函数宽度
4. fn_call_width = 90
5. # 链式调用一行最大宽度
6. chain_one_line_max = 80
7. # 压缩通配符前缀
8. condense wildcard suffixes = true

rustfmt

Cargo cargo-fix cargo-clippy
cargo-fix cargo fix Warning

cargo-clippy Rust 1.29 cargo clippy Rust

10.2

Rust crate src/lib.rs Rust

mod static_hashmap static_kv Rust top-level src/lib.rs src/main.rs

Rust 2015

static_hashmap Rust 2015 src/main.rs static_kv static_kv.rs 10-34

10-34

```
src/
|   ├── main.rs
|   └── static_kv.rs
```

main.rs mod static_kv{...} static_kv.rs mod 10-35

10-35 static_kü static_kü.rs

```

1. // src/static_kv.rs
2. use std::collections::HashMap;
3. use std::sync::RwLock;
4. pub const NF: &'static str = "not found";
5. lazy_static! {
6.     pub static ref MAP: HashMap<u32, &'static str> = {
7.         let mut m = HashMap::new();
8.         m.insert(0, "foo");
9.         m
10.    };
11.    pub static ref MAP_MUT: RwLock<HashMap<u32, &'static str>> =
12.    {
13.        let mut m = HashMap::new();
14.        m.insert(0, "bar");
15.        RwLock::new(m)
16.    };
17. }

```

10-35 static_kv.rs mod
 Cargo static_kv.rs static_kv mod
 main.rs static_kv
 static_kv MAP &static_kv static_kv NF

main.rs static_kv.rs mod
 static_kv 10-36

10-36 main.rs mod static_kü

```

1. // main.rs
2. #[macro_use]
3. extern crate lazy_static;
4. mod static_kv;
5. fn read_kv() {
6.     // ...
7. }
8. fn rw_mut_kv() -> Result<(), String> {
9.     // ...
10. }
11. fn main() {
12.     // ...
13. }

```

10-36 4 mod static_kv Cargo
static_kv.rs

main.rs read_kv rw_mut_kv
src read_func.rs
10-37

10-37 read_kü rw_mut_kü main.rs
read_func.rs

```

1. // read_func.rs
2. use static_kv;
3. pub fn read_kv() {
4.     let ref m = static_kv::MAP;
5.     assert_eq!("foo", *m.get(&0).unwrap_or(&static_kv::NF));
6.     assert_eq!(
7.         static_kv::NF, *m.get(&1).unwrap_or(&static_kv::NF)
8.     );
9. }
10. pub fn rw_mut_kv() -> Result<(), String> {
11.     {
12.         let m = static_kv::MAP_MUT
13.             .read().map_err(|e| e.to_string())?;
14.         assert_eq!("bar", *m.get(&0).unwrap_or(&static_kv::NF));
15.     }
16.     {
17.         let mut m = static_kv::MAP_MUT
18.             .write().map_err(|e| e.to_string())?;
19.         m.insert(1, "baz");
20.     }
21.     Ok(())
22. }

```

10-37 read_kv rw_mut_kv

read_func.rs

static_kv

static_kv main.rs mod

read_func.rs use 2

read_kv rw_mut_kv pub

public

pub 3 10

main.rs mod read_func 10-38

10-38 main.rs read_func

```

1. // main.rs
2. #[macro_use]
3. extern crate lazy_static;
4. mod static_kv;
5. mod read_func;
6. use read_func::{read_kv, rw_mut_kv};
7. fn main() {
8.     // ...
9. }

```

10-38 main.rs 5 mod read_func 6 use read_func read_kv rw_mut_kv main read_func read_kv read_func rw_mut_kv

main.rs cargo run

src main.rs read_func.rs static_kv.rs static_kv.rs read_func.rs MAP MAP_MUT static_kv.rs read_func.rs 10-39

10-39 static_kv.rs read_func.rs

```

static_hashmap
├── src
│   ├── main.rs
│   └── static_func
│       ├── mod.rs
│       ├── read_func.rs
│       └── static_kv.rs

```

10-39 static_func static_kv.rs read_func.rs mod.rs static_func Cargo mod.rs static_kv read_func 10-40

10-40 mod.rs static_kv read_func

```
1. // src/static_func/mod.rs
2. pub mod static_kv;
3. pub mod read_func;
```

10-40 mod static_kv read_func

```
pub
```

cargo build

```
error[E0432]: unresolved import `static_kv`
--> src/static_func/read_func.rs:1:5
1 | use static_kv;
  |     ^^^^^^^^^ no `static_kv` in the root
```

read_func.rs use static_kv root

static_kv Cargo use static_kv use super static_kv super Cargo super

main.rs 10-41

10-41 main.rs

```
1. // main.rs
2. #[macro_use]
3. extern crate lazy_static;
4. mod static_func;
5. use static_func::static_kv;
6. use static_func::read_func::{read_kv, rw_mut_kv};
7. fn main() {
8.     // ...
9. }
```

10-41 4 mod static_func 5

6 use static_func static_kv static_func read_func main main

Cargo.toml edition "2015" cargo run

static_hashmap_2015

read_func.rs mod static_kv

10-43

10-43 read_func.rs static_kv

```
1. pub mod static_kv;
2. pub fn read_kv() {
3.     // 同 static_hashmap_2015 项目内容
4. }
5. pub fn rw_mut_kv() -> Result<(), String> {
6.     // 同 static_hashmap_2015 项目内容
7. }
```

10-43 read_kv rw_mut_kv static_hashmap_2015 1 static_kv Rust mod read_func static_kv

static_kv.rs 10-44

10-44 read_func/static_kv.rs

```
1. use lazy_static::lazy_static;
2. use std::collections::HashMap;
3. use std::sync::RwLock;
4. pub const NF: &'static str = "not found";
5. lazy_static! {
6.     // 同 static_hashmap_2015 项目内容
7. }
```

10-44 lazy_static 1 use lazy_static lazy_static Rust 2018 main.rs [macro_use] extern crate lazy_static lazy_static use

main.rs 10-45

10-45 main.rs

```

1. mod read_func;
2. use crate::read_func::{read_kv, rw_mut_kv};
3. fn main() {
4.     read_kv();
5.     match rw_mut_kv() {
6.         Ok(()) => {
7.             let m = read_func::static_kv::MAP_MUT
8.                 .read()
9.                 .map_err(|e| e.to_string()).unwrap();
10.            assert_eq!("baz",
11.                *m.get(&1).unwrap_or(&read_func::static_kv::NF)
12.            );
13.        }
14.        Err(e) => println!("Error {}", e),
15.    }
16. }

```

10-45 mod read_func static_hashmap_2015 main.rs 7 static_kv MAP_MUT read_func

2 use crate crate read_func self crate Rust main.rs read_func crate

10.3

Cargo 2017 C++17 Rust

CSV 10-46

10-46 CSV


```

csv_challenge
├── Cargo.toml
├── input
│   └── challenge.csv
├── output/
└── src
    └── main.rs

```

std::env::args() を使って、command line arguments を取得する。

10.3.2 structopt を使う

structopt は clap を使って、command line arguments を取得するためのライブラリです。Cargo.toml に以下を追加します。

1. [dependencies]
2. structopt = "0.2"
3. structopt-derive = "0.2"

cargo build crates.io から structopt をインストールします。structopt-derive は procedural macro なので、12.1.1 で説明するように、src/main.rs の最上段に以下を追加します。

structopt を使って、command line arguments を取得する。src/main.rs の最上段に以下を追加します。

```

csv_challenge
├── Cargo.toml
├── input
│   └── challenge.csv
├── output/
└── src
    ├── opt.rs
    └── main.rs

```

opt.rs の内容 (Opt は 10-48 ページ参照)

10-48 src/opt.rs 10-48 Opt

```
1. use structopt_derive::*;
2. #[derive(StructOpt, Debug)]
3. #[structopt(name = "csv_challenge", about = "Usage")]
4. pub struct Opt {
5.     #[structopt(help = "Input file")]
6.     pub input: String,
7.     #[structopt(help = "Column Name")]
8.     pub column_name: String,
9.     #[structopt(help = "Replacement Column Name")]
10.    pub replacement: String,
11.    #[structopt(help = "Output file, stdout if not present")]
12.    pub output: Option<String>,
13. }
```

10-48 10-48 Opt 10-48

USAGE:

csv_challenge [FLAGS] <input> <column_name> <replacement> [output]

10-48

- **csv_challenge** 10-48
 - **[FLAGS]** 10-48 Flag 10-48 " " " " 10-48
 - **input** 10-48 CSV 10-48 6 10-48 input 10-48 [structopt...] 10-48 help 10-48
 - **column_name** 10-48 CSV 10-48 8 10-48 column_name 10-48
 - **replacement** 10-48 10 10-48 replacement 10-48
 - **[output]** 10-48 12 10-48 output 10-48 Option<String> 10-48 10-48
- 10-48 opt 10-48 10-48 pub 10-48 10-48

10-48 src/main.rs 10-48 10-49 10-48

10-49 src/main.rs

```
1. use structopt::StructOpt;
2. mod opt;
3. use self::opt::Opt;
4. fn main() {
5.     let opt = Opt::from_args();
6.     println!("{:?}", opt);
7. }
```

10-49 Rust 2018 extern crate
[macro_use] structopt structopt_derive use
StructOpt

2 3 mod use Opt main
structopt Opt from_args cargo
run

error: The following required arguments were not provided:

<input>

<column_name>

<replacement>

USAGE:

csv_challenge [FLAGS] <input> <column_name> <replacement> [output]

For more information try --help

structopt
--help cargo run---help

USAGE:

csv_challenge [FLAGS] <input> <column_name> <replacement> [output]

FLAGS:

-h, --help Prints help information
-V, --version Prints version information
-v, --verbose

ARGS:

<input> Input file
<column_name> Column Name
<replacement> Replacement Column Name
<output> Output file, stdout if not present

structopt **Opt** **csv_challenge**

10.3.3

src **src/err.rs**

10-50 **src/err.rs**

1. use std::io;
2. #[derive(Debug)]
3. pub enum Error {

```

4.      Io(io::Error),
5.      Program(&'static str),
6.  }
7.  impl From<io::Error> for Error {
8.      fn from(e: io::Error) -> Error {
9.          Error::Io(e)
10.     }
11. }
12. impl From<&'static str> for Error {
13.     fn from(e: &'static str) -> Error {
14.         Error::Program(e)
15.     }
16. }

```

10-50 行代码实现 Error 的 Io 和 Program 的 I/O 操作。
 From 的实现 I/O 操作。
 src/main.rs 中实现 csv_challenge。

```

csv_challenge
├── Cargo.toml
├── input
│   ├── challenge.csv
├── output/
└── src
    ├── err.rs
    ├── opt.rs
    └── main.rs

```

CSV 操作

10.3.4 CSV

src 目录下的 core.rs 实现 CSV 操作。
 CSV 操作。
 core/read.rs 和 core/write.rs 实现 csv_challenge。


```

csv_challenge
├── Cargo.toml
├── input
│   ├── challenge.csv
├── output/
└── src
    ├── core
    │   ├── read.rs
    │   └── write.rs
    ├── core.rs
    ├── err.rs
    ├── opt.rs
    └── main.rs

```

□□□□□core.rs□□□□□□□□10-51□□□

□□□□10-51□□□core.rs □□

1. pub mod read;
2. pub mod write;
3. use crate::err::Error;

□□□□□ pub □□□□□□□□□□□□□□□□ main □□□□□□□□□□□□□□□□

□read□□□□□Error□□□□□□core.rs□□□□□□□

□□□□core/read.rs□□□□□□□□□□□□□□□□10-52□□□

□□□□10-52□□core/read.rs□□□□□□□□□□□

```

1. use std::path::PathBuf;
2. use std::fs::File;
3. use super::Error;
4. use std::io::prelude::*;
5. pub fn load_csv(csv_file: PathBuf) -> Result<String, Error> {
6.     let file = read(csv_file)?;
7.     Ok(file)
8. }
9. pub fn write_csv(csv_data: &str, filename: &str) -> Result<(), Error>
10. {
11.     write(csv_data, filename)?;
12.     Ok(())
13. }
14. fn read(path: PathBuf) -> Result<String, Error> {
15.     let mut buffer = String::new();
16.     let mut file = open(path)?;
17.     file.read_to_string(&mut buffer)?;
18.     if buffer.is_empty() {
19.         return Err("input file missing");
20.     }
21.     Ok(buffer)
22. }
23. fn open(path: PathBuf) -> Result<File, Error> {
24.     let file = File::open(path)?;
25.     Ok(file)
26. }
27. fn write(data: &str, filename: &str) -> Result<(), Error> {
28.     let mut buffer = File::create(filename)?;
29.     buffer.write_all(data.as_bytes())?;
30.     Ok(())
31. }

```

10-52 1 4 use std::path::PathBuf std::fs::File std::io::prelude::*
 Rust extern crate std use std

```
prelude v1 * use std
prelude v1 * [4] 3 super
core Error
```

```
Rust Path PathBuf
&str String Path PathBuf
```

```
std fs File open create
CSV
```

```
std io I/O trait Read Write Seek BufRead trait I/O std io
prelude * I/O
```

```
5 13 pub load_csv write_csv
CSV read open write
```

```
main.rs CSV 10-53
10-53 main.rs load_cs write_cs CSV
```

```

1. use structopt::StructOpt;
2. mod opt;
3. use opt::Opt;
4. mod err;
5. mod core;
6. use self::core::read::{load_csv, write_csv};
7. use std::path::PathBuf;
8. use std::process;
9. fn main() {
10.     let opt = Opt::from_args();
11.     let filename = PathBuf::from(opt.input);
12.     let csv_data = match load_csv(filename) {
13.         Ok(fname) => { fname },
14.         Err(e) => {
15.             println!("main error: {:?}", e);
16.             process::exit(1);
17.         }
18.     };
19.     let output_file = &opt.output
20.         .unwrap_or("output/output.csv".to_string());
21.     match write_csv(&csv_data, &output_file) {
22.         Ok(_) => {
23.             println!("write success!");
24.         },
25.         Err(e) => {
26.             println!("main error: {:?}", e);
27.             process::exit(1);
28.         }
29.     }
30. }

```

10-53

11 PathBuf from opt.input CSV
PathBuf

```
1218fn load_csv(filename: &str) -> Result<Vec<str>, Error> {
    match load_csv(filename) {
        Ok(result) => Ok(result),
        Err(e) => Err(e),
    }
}
```

```
1920fn output_csv(filename: &str, data: Vec<str>) -> Result<Vec<str>, Error> {
    opt.output_csv(filename, data, unwrap_or_default)
}
```

```
2129fn write_csv(filename: &str, data: Vec<str>) -> Result<Vec<str>, Error> {
    write_csv(filename, data, unwrap_or_default)
}
```

```
cargo run input/challenge.csv City Beijing --output/output.csv
```

```
core/write.rs
```

10.3.5 CSV

```
lines: Vec<str> -> Result<Vec<str>, Error> {
    CSV::from_lines(lines)
}
```

```
PathBuf, File, Error
```

```
core.rs
```

```
10-54 core.rs core/read.rs
```

```
1. // core.rs
2. pub mod read;
3. pub mod write;
4. use crate::err::Error;
5. use std::{
6.     path::PathBuf,
7.     fs::File,
8.     io::{Read, Write},
9. };
10. // core/read.rs
11. use super::{Error, PathBuf, File, Read, Write};
```

10-54 core/read.rs std Rust
2018 use core.rs core/read.rs
super

core/write.rs 10-55

10-55 core/write.rs

```
1. use super::*;
2. pub fn replace_column(data: String, column: &str, replacement: &str)
3.     -> Result<String, Error> {
4.     let mut lines = data.lines();
5.     let headers = lines.next().unwrap();
6.     let columns: Vec<&str> = headers.split(',').collect();
7.     let column_number = columns.iter().position(|&e| e == column);
8.     let column_number = match column_number {
9.         Some(column) => column,
10.        None => Err("column name doesn't exist in the input file")?
11.    };
12.    let mut result = String::with_capacity(data.capacity());
13.    result.push_str(&columns.join(","));
14.    result.push('\n');
15.    for line in lines {
16.        let mut records: Vec<&str> = line.split(',').collect();
17.        records[column_number] = replacement;
18.        result.push_str(&records.join(","));
19.        result.push('\n');
20.    }
21.    Ok(result)
22. }
```

10-55 1 use spuer * core

2 pub replace_column CSV
data column replacement

```

    411 lines = data.next().CSV
    headers = split headers Vec&str
    columns = position column_number
    match position

```

```

    1220 result = columns.join
    lines = Vec&str
    records = column_number
    replacement = records.join

```

```

    21

```

```

    main replace_column 10-56

```

```

    10-56 main. replace_column

```

```

1. // .....
2. use self::core::{
3.     read::{load_csv, write_csv},
4.     write::replace_column,
5. };
6. fn main() {
7.     // .....
8.     let modified_data = match
9.     replace_column(csv_data, &opt.column_name, &opt.replacement)
10.    {
11.        Ok(data) => { data },
12.        Err(e) => {
13.            println!("main error: {:?}", e);
14.            process::exit(1);
15.        }
16.    };
17.    // .....
18.    match write_csv(&modified_data, &output_file) {
19.        // .....
20.    }
21. }

```

10-56 main.rs 25 use core
read write

8 16 replace_column CSV
modified_data

18 modified_data output_file

cargo run input/challenge.csv City Beijing
output/output.csv City Beijing

10.3.6

csv_challenge Rust
mod test load_csv

10-57

10-57 load_csv

```
1. #[cfg(test)]
2. mod test {
3.     use std::path::PathBuf;
4.     use super::load_csv;
5.     #[test]
6.     fn test_valid_load_csv() {
7.         let filename = PathBuf::from("./input/challenge.csv");
8.         let csv_data = load_csv(filename);
9.         assert!(csv_data.is_ok());
10.    }
11. }
```

10-57 1 [cfg test] cargo test

2 test use PathBuf load_csv
PathBuf load_csv test

第 5 章 **[test]** 测试 **[ignore]** 忽略

第 6 章 并行编程

第 7 章 cargo test 测试 `load_csv` `write_csv` `replace_column` `csv_challenge`

第 8 章

第 9 章 Rust 的 `csv_challenge`

第 10 章 `src/lib.rs` `main.rs` 10-58

第 10-58 章 `src/lib.rs`

```
1. mod opt;
2. mod err;
3. mod core;
4. // 重新导出
5. pub use self::opt::Opt;
6. pub use self::core::{
7.     read::{load_csv, write_csv},
8.     write::replace_column,
9. }
```

第 10-58 章 `opt` `err` `core`

第 5 章 9 章 `Re-exporting`

第 10 章 `mian.rs` 10-59

第 10-59 章 `main.rs`

```

1. // .....
2. use structopt::StructOpt;
3. use csv_challenge::{
4.     Opt,
5.     {load_csv, write_csv},
6.     replace_column,
7. };
8. use std::path::PathBuf;
9. use std::process;
10. // .....
11. fn main() {
12.     // .....
13. }

```

10-59 main.rs 3
 csv_challenge

main.rs lib.rs

cargo build csv_challenge
 tests integration_test.rs input
 CSV no_header.csv CSV
 csv_challenge

```

csv_challenge
├── Cargo.toml
├── input
│   ├── challenge.csv
│   └── no_header.csv
├── output/
└── src
    ├── core.rs
    │   ├── read.rs
    │   └── write.rs
    ├── err.rs
    ├── opt.rs
    ├── lib.rs
    ├── main.rs
    └── tests
        └── integration_test.rs

```

10-60 integration_test.rs
 10-60 integration_test.rs

```

1.  #[cfg(test)]
2.  mod test {
3.      use std::path::PathBuf;
4.      use csv_challenge::{
5.          Opt,
6.          {load_csv, write_csv},
7.          replace_column,
8.      };
9.      #[test]
10.     fn test_csv_challenge(){
11.         let filename = PathBuf::from("./input/challenge.csv");
12.         let csv_data = load_csv(filename).unwrap();
13.         assert!(csv_data.is_ok());
14.         let modified_data = replace_column(
15.             csv_data, "City", "Beijing"
16.         ).unwrap();
17.         assert!(modified_data.is_ok());
18.         let output_file = write_csv(
19.             &modified_data, "output/test.csv"
20.         );
21.         assert!(output_file.is_ok());
22.     }
23. }

```

10-61 Rust 10-61 core/read.rs write_csv

```

1.  /// # Usage:
2.  /// ```ignore
3.  /// let filename = PathBuf::from("./files/challenge.csv");
4.  /// let csv_data = load_csv(filename).unwrap();
5.  /// let modified_data = replace_column(
6.  ///     csv_data, "City", "Beijing").unwrap();
7.  /// let output_file = write_csv(&modified_data, "output/test.csv");
8.  /// assert!(output_file.is_ok());
9.  /// ```
10. pub fn write_csv(csv_data: &str, filename: &str)
11. -> Result<(), Error>
12. {
13.     write(csv_data, filename)?;
14.     Ok(())
15. }

```

10-61 `write_csv` // **Markdown** `Rust` `///`

2 `ignore` `cargo test` `Rust` ````` `write_csv` ````ignore`

`///` 10-62

10-62 `src/lib.rs`

```

1.  ///! This is documentation for the `csv_challenge` lib crate
2.  ///!
3.  ///! Usage:
4.  ///! ```
5.  ///!     use csv_challenge::{
6.  ///!         Opt,
7.  ///!         {load_csv, write_csv},
8.  ///!         replace_column,
9.  ///!     };
10. ///! ```

```

10-62 src/lib.rs “// ”

cargo test

cargo doc target/doc UI

Rust csv_challenge **benches**

Cargo

benches/file_op_bench.rs 10-63

10-63 benches/file_op_bench.rs

```

1.  #![feature(test)]
2.  extern crate test;
3.  use test::Bencher;
4.  use std::path::PathBuf;
5.  use csv_challenge::{
6.      Opt,
7.      {load_csv, write_csv},
8.      replace_column,
9.  };
10. #[bench]
11. fn bench_read_100times(b: &mut Bencher) {
12.     b.iter(|| {
13.         let n = test::black_box(100);
14.         (0..n).fold(0, |_,_|{test_load_csv();0})
15.     });
16. }
17. fn test_load_csv(){
18.     let filename = PathBuf::from("./input/challenge.csv");
19.     load_csv(filename);
20. }

```

11-16行是 **features** 中的 **test** 模块，是 Rust 中的 **features** 模块，`extern crate test` 是 `test` 模块，`test::Bencher` 是 `iter` 模块。

11-16行是 `bench_read_100times` 函数，是 **bench** 模块，`test::black_box` 是 `100` 模块，`test_load_csv` 是 `100` 模块，`load_csv` 是 `fold` 模块，`test_load_csv` 是 `100` 模块，`14` 是 `14` 模块。

`cargo bench` 是 `cargo` 模块。

`test bench_read_100times ... bench: 1,321,230 ns/iter (+/- 699,240)`

`crates.io`

crates.io crates.io
 Api Token Token .cargo/config
 [registry] cargo login
 Token

cargo publish crates.io
 cargo package
 target/package

cargo publish

error: api errors: missing or empty metadata fields: description, license.
Please see <http://doc.crates.io/manifest.html#package-metadata> for how to
upload metadata

Cargo.toml [package]
 description license
 cargo install

10.4

Rust Enum
 Item pub
 pub

10-64 Rust 2015

10-64 Rust 2015


```
1. pub mod outer_mod {
2.     pub(self) fn outer_mod_fn() {}
3.     pub mod inner_mod {
4.         // 对外层模块 `outer_mod` 可见
5.         pub(in outer_mod) fn outer_mod_visible_fn() {}
6.         // 对整个 crate 可见
7.         pub(crate) fn crate_visible_fn() {}
8.         // 在 `outer_mod` 内部可见
9.         pub(super) fn super_mod_visible_fn() {
10.            // 访问同一个模块的函数
11.            inner_mod_visible_fn();
12.            // 访问父模块的函数需要使用 "::" 前缀
13.            ::outer_mod::outer_mod_fn();
14.        }
15.        // 仅在 `inner_mod` 内部可见
16.        pub(self) fn inner_mod_visible_fn() {}
17.    }
18.    pub fn foo() {
19.        inner_mod::outer_mod_visible_fn();
20.        inner_mod::crate_visible_fn();
21.        inner_mod::super_mod_visible_fn();
22.        // 不能使用 inner_mod 的私有函数
23.        // inner_mod::inner_mod_visible_fn();
24.    }
25. }
26. fn bar() {
27.    // 该函数对整个 crate 可见
28.    outer_mod::inner_mod::crate_visible_fn();
29.    // 该函数只对 outer_mod 可见
30.    // outer_mod::inner_mod::super_mod_visible_fn();
31.    // 该函数只对 outer_mod 可见
32.    // outer_mod::inner_mod::outer_mod_visible_fn();
33.    // 通过 foo 函数调用内部细节
```

```

34.     outer_mod::foo();
35. }
36. fn main() { bar() }

```

10-64 outer_mod inner_mod
outer_mod crate

2 pub self outer_mod_fn self
outer_mod inner_mod

5 pub in outer_mod
outer_mod_visible_fn outer_mod
inner_mod outer_mod

7 pub crate crate_visible_fn
crate

9 pub super super_mod_visible_fn
outer_mod pub in outer_mod super

13 super_mod_visible_fn outer_mod
outer_mod_fn " " " " " "
outer_mod " outer_mod
outer_mod_fn Rust **Uniform Path**

16 pub self inner_mod
inner_mod_visible_fn inner_mod

18 24 pub foo inner_mod
inner_mod_visible_fn
inner_mod pub foo

26 35 bar outer_mod
— crate crate_visible_fn foo

- pub
- pub

- `pub crate` 与 `crate`
- `pub in Path` 与 `Path` 与 `Path`
- `pub self` 与 `pub in self`
- `pub super` 与 `pub in super`

模块系统

模块 **Rust 2018** 模块系统 10-65

□□

模块 **10-65 Rust 2018**

```

1. pub mod outer_mod {
2.     pub(self) fn outer_mod_fn() {}
3.     pub mod inner_mod {
4.         // 在 Rust 2018 模块系统中必须使用 use 导入
5.         use crate::outer_mod::outer_mod_fn;
6.         // 对外层模块 `outer_mod` 可见
7.         pub(in crate::outer_mod) fn outer_mod_visible_fn() {}
8.         // 在 `outer_mod` 内部可见
9.         pub(super) fn super_mod_visible_fn() {
10.            // 访问同一个模块的函数
11.            inner_mod_visible_fn();
12.            // 因为使用 use 导入了 outer_mod, 所以这里直接使用
13.            outer_mod_fn();
14.        }
15.        // 其他代码同上
16.    }
17.    pub fn foo() {
18.        // 代码同上
19.    }
20. }
21. // 其他代码同上

```

模块 10-65 Rust 2018 模块系统 Rust 2015

模块系统

如何安装 Rust 的 TOML 支持

-
- [1] 在 [GitHub](#) 上找到 TOML 0.4.0 的仓库 [toml-lang/toml](#)
 - [2] 查看该仓库的 README 文件 <https://semver.org>
 - [3] 在 [GitHub](#) 上找到 [rust-lang/regex](#) 仓库
 - [4] 在 <https://doc.rust-lang.org/std/prelude/index.html> 上找到

11 并行

并行编程模型

并行编程模型是指一种编程模型，它允许程序员将程序分解成多个可以同时执行的子程序。这种模型通常用于提高程序的执行效率，特别是在处理大规模数据或复杂计算时。并行编程模型可以分为共享内存模型和分布式内存模型两大类。

共享内存模型是指所有处理器共享同一块内存，每个处理器通过访问共享内存来与其他处理器进行通信。这种模型的优点是编程简单，但缺点是容易出现数据竞争和死锁等问题。分布式内存模型是指每个处理器拥有自己的私有内存，处理器之间通过消息传递的方式进行通信。这种模型的优点是易于扩展，但缺点是编程复杂。

并行编程模型的应用非常广泛，特别是在高性能计算、大数据处理和嵌入式系统等领域。随着硬件技术的发展，并行编程模型的重要性日益凸显。App 并行编程模型是一种基于消息传递的并行编程模型，它允许程序员将程序分解成多个可以同时执行的子程序，每个子程序通过消息传递与其他子程序进行通信。

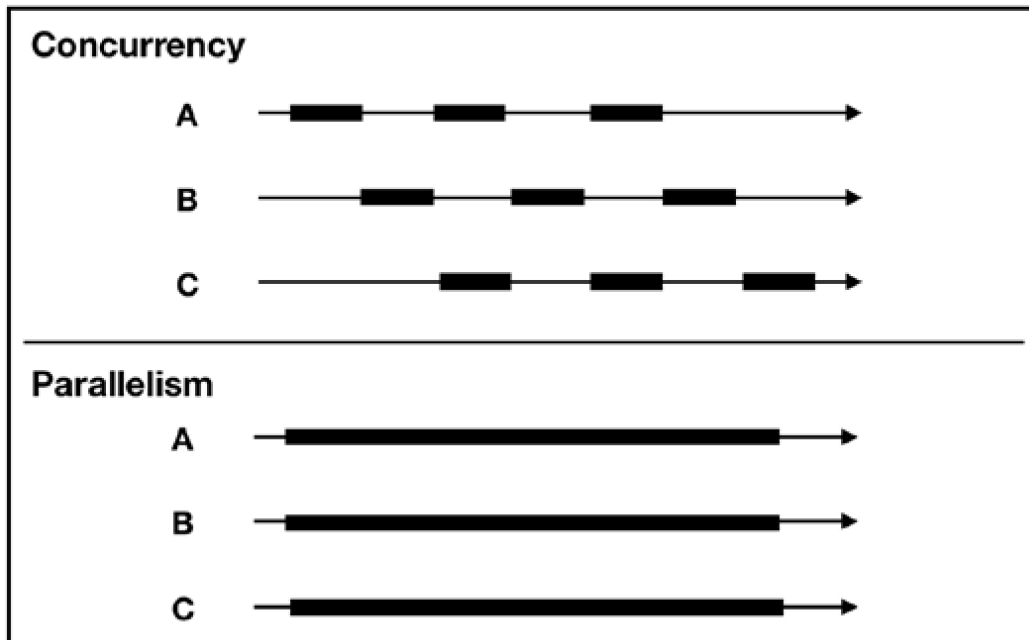
11.1 并行

并行（Concurrency）是指同时执行多个任务的能力。并行与并行（Parallelism）不同，并行是指同时执行多个任务，而并行是指同时执行多个任务。并行编程模型是指一种编程模型，它允许程序员将程序分解成多个可以同时执行的子程序。

并行编程模型的应用非常广泛，特别是在高性能计算、大数据处理和嵌入式系统等领域。随着硬件技术的发展，并行编程模型的重要性日益凸显。Rob Pike 在其著作《Dealing With Doing》中提出了并行编程模型的概念，他认为并行编程模型是一种可以同时执行多个任务的编程模型。

并行编程模型的应用非常广泛，特别是在高性能计算、大数据处理和嵌入式系统等领域。随着硬件技术的发展，并行编程模型的重要性日益凸显。并行编程模型是指一种编程模型，它允许程序员将程序分解成多个可以同时执行的子程序。并行编程模型可以分为共享内存模型和分布式内存模型两大类。

11-1 并行编程模型



11-1 並行性と並列性

並行性と並列性は、どちらも複数のタスクを同時に実行することを指しますが、その実行方法が異なります。

並行性は、複数のタスクが同時に実行されることを指します。

並列性は、複数のタスクが同時に実行されることを指しますが、その実行方法が異なります。Go言語は、並列性をサポートする言語です。

並行性は、複数のタスクが同時に実行されることを指します。

- ・ 並行性は、複数のタスクが同時に実行されることを指します。
- ・ 並列性は、複数のタスクが同時に実行されることを指します。

11.1.1 並行性

並行性は、複数のタスクが同時に実行されることを指します。

並列性は、複数のタスクが同時に実行されることを指しますが、その実行方法が異なります。CPUは、並列性をサポートするハードウェアです。

Master-Worker 模式中 Master 负责 Worker 的 Worker 通过 Worker 与 Master 通过 Socket 或 IPC 进行通信。Worker 负责具体的业务逻辑，CPU 负责具体的业务逻辑。

Master 负责 Worker 的 Worker 通过 Worker 与 Master 通过 Socket 或 IPC 进行通信。Worker 负责具体的业务逻辑，CPU 负责具体的业务逻辑。

11.1.2 异步编程

C10K 问题是指 10K 个问题。C10K 问题是指 10K 个问题。

C10K 问题是指 10K 个问题。Linux 中的 **epoll** 是解决 C10K 问题的方案。Node.js 中的 **Callback** 是解决 C10K 问题的方案。

C10K 问题是指 10K 个问题。Call Hell 是指 10K 个问题。

Promise 和 Future 是解决 C10K 问题的方案。Promise 和 Future 是解决 C10K 问题的方案。

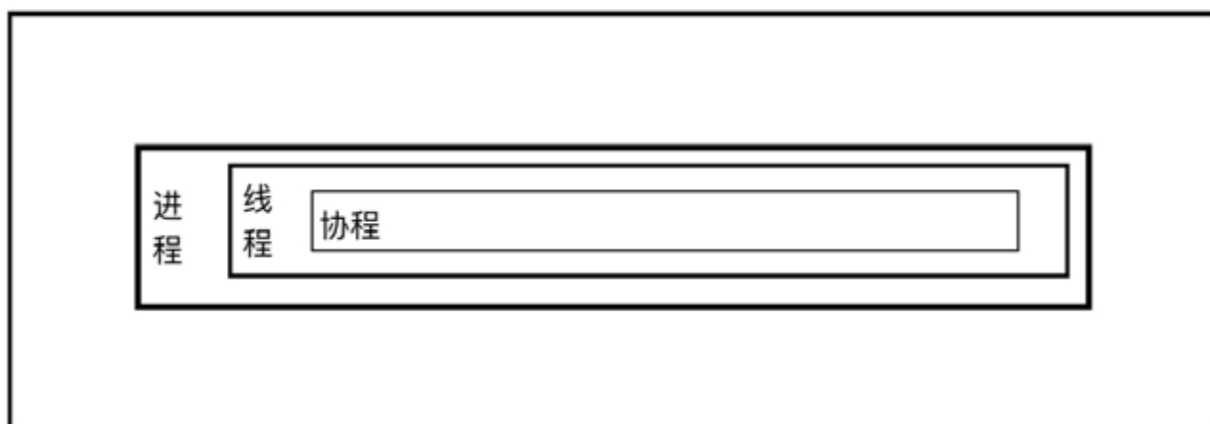
Promise/Future 是 JavaScript 中实现异步编程的常用工具，它们提供了一种更简洁、更直观的方式来处理异步操作。

在 20 世纪 60 年代，COBOL 是当时最流行的编程语言之一，它主要用于处理大规模的数据处理任务。

在计算机科学中，进程和线程是两个重要的概念。进程是一个正在运行的程序，而线程是进程中的一个执行单元。多线程编程可以提高程序的并发性和效率，但同时也带来了同步和通信的问题。

Go 语言中的 goroutines 和 Erlang 语言中的 LWP 都是实现多线程编程的工具。Python 和 Ruby 也有类似的概念，如 Python 的 threads 和 Ruby 的 threads。JavaScript 中的 Promise 和 Future 则是处理异步操作的常用工具。Rust 语言则提供了一种更安全的多线程编程方式。

11-2 多线程编程



11-2 多线程编程

多线程编程可以提高程序的并发性和效率，但同时也带来了同步和通信的问题。在多线程编程中，需要特别注意线程的安全性和同步问题。

11.1.3 测试

测试代码在 `src/main.rs` 文件中，我们使用 `println!` 来输出一些信息，[\[1\]](#) 我们使用 `assert_eq!` 来断言一些值是否相等，`Bug` 是一个宏，用于测试一些 Bug。

我们使用 `println!` 来输出一些信息，我们使用 `assert_eq!` 来断言一些值是否相等，我们使用 `Bug` 来测试一些 Bug。

测试代码

我们使用 `println!` 来输出一些信息，我们使用 `assert_eq!` 来断言一些值是否相等，我们使用 `Bug` 来测试一些 Bug。

测试

我们使用 `println!` 来输出一些信息，我们使用 `assert_eq!` 来断言一些值是否相等，我们使用 `Bug` 来测试一些 Bug。

```
1. static mut V: i32 = 0;
2. fn unsafe_seq() -> i32{
3.     unsafe{
4.         V += 1;
5.         V
6.     }
7. }
```

我们使用 `println!` 来输出一些信息，我们使用 `assert_eq!` 来断言一些值是否相等，我们使用 `Bug` 来测试一些 Bug。

我们使用 `println!` 来输出一些信息，我们使用 `assert_eq!` 来断言一些值是否相等，我们使用 `Bug` 来测试一些 Bug。

我们使用 `println!` 来输出一些信息，我们使用 `assert_eq!` 来断言一些值是否相等，我们使用 `Bug` 来测试一些 Bug。

我们使用 `println!` 来输出一些信息，我们使用 `assert_eq!` 来断言一些值是否相等，我们使用 `Bug` 来测试一些 Bug。

我们使用 `println!` 来输出一些信息，我们使用 `assert_eq!` 来断言一些值是否相等，我们使用 `Bug` 来测试一些 Bug。

假设初始时V=0，线程A执行2次，线程B执行1次，线程A执行3次。
 “V+=1”操作分解为11-3步

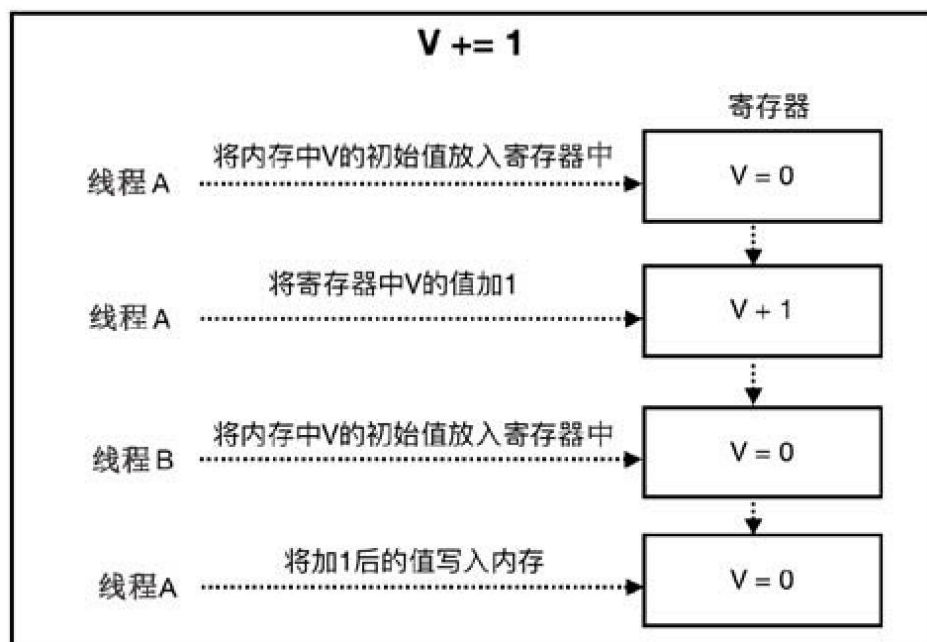


图11-3 “V+=1”操作分解

图11-1展示了多线程环境下，多个线程同时访问共享变量V时，由于缺乏同步机制，导致数据不一致的问题。这种现象被称为**Race Condition**（竞态条件）。

图11-1展示了多线程环境下，多个线程同时访问共享变量V时，由于缺乏同步机制，导致数据不一致的问题。这种现象被称为**Data Race**（数据竞争）。

图11-1展示了多线程环境下，多个线程同时访问共享变量V时，由于缺乏同步机制，导致数据不一致的问题。

图11-1展示了多线程环境下，多个线程同时访问共享变量V时，由于缺乏同步机制，导致数据不一致的问题。

11-2
trans1

11-2

```
1. trans1(amount, account_from, account_to){
2.     if (account_from.balance < amount) return FALSE;
3.     account_to.balance += amount;
4.     account_from.balance -= amount;
5.     return TRUE
6. }
```

Mutex
11-3

11-3

```
1. trans2(amount, account_from, account_to){
2.     atomic { bal = account_from.balance; }
3.     if (bal < amount) return FALSE;
4.     atomic { account_to.balance += amount; }
5.     atomic { account_from.balance -= amount; }
6.     return TRUE;
7. }
```

11-3 atomic
trans2
11-4

11-4

```

1. trans3(amount, account_from, account_to){
2.     atomic {
3.         if (account_from.balance < amount) return FALSE;
4.         account_to.balance += amount;
5.         account_from.balance -= amount;
6.         return TRUE;
7.     }
8. }

```

trans3 関数は atomic 関数を用いて、
 同時に実行される複数の関数を安全に実行するための関数。
 11-5 図
 11-5 図

```

1. trans4(amount, account_from, account_to){
2.     account_from.activity = true;
3.     account_to.activity = true;
4.     atomic {
5.         if (account_from.balance < amount) return FALSE;
6.         account_to.balance += amount;
7.         account_from.balance -= amount;
8.         return TRUE;
9.     }
10. }

```

trans4 関数は 2 つの関数を同時に実行するための関数。
 11-6 図

11-6 図

Rust 関数 unsafe_seq 関数
 11-6 図 unsafe_seq 関数

```

1. use std::thread;
2. static mut V: i32 = 0;
3. fn unsafe_seq() -> i32 {
4.     unsafe {
5.         V += 1;
6.         V
7.     }
8. }
9. fn main() {
10.    let child = thread::spawn(move || {
11.        for _ in 0..10 {
12.            unsafe_seq();
13.            unsafe{println!("child : {}", V);}
14.        }
15.    });
16.    for _ in 0..10 {
17.        unsafe_seq();
18.        unsafe{println!("main : {}", V);}
19.    }
20.    child.join().unwrap();
21. }

```

11-6 std::thread::spawn main
 child unsafe_seq 10 15 main
 unsafe_seq 16 19 20 child
 join

main child
 0 20

- main 0 10
- child main

main child
 0 20

main child
 0 20

操作系统中，线程同步是保证程序正确执行的关键。在多线程环境中，多个线程可能会同时访问共享资源，如果没有适当的同步机制，就会导致数据不一致或程序崩溃。本章将介绍几种常见的线程同步方法，包括互斥锁、信号量、屏障、条件变量、读写锁和自旋锁。

本章将介绍几种常见的线程同步方法，包括互斥锁、信号量、屏障、条件变量、读写锁和自旋锁。互斥锁（Mutex）是最基本的同步机制，用于保证同一时刻只有一个线程可以访问共享资源。读写锁（RwLock）允许多个线程同时读取共享资源，但在写入时必须互斥。自旋锁（Spinlock）适用于处理器缓存一致性的系统，通过不断检查锁的状态来实现同步。信号量（Semaphores）和屏障（Barrier）则用于更复杂的同步场景，如控制多个线程的执行顺序。

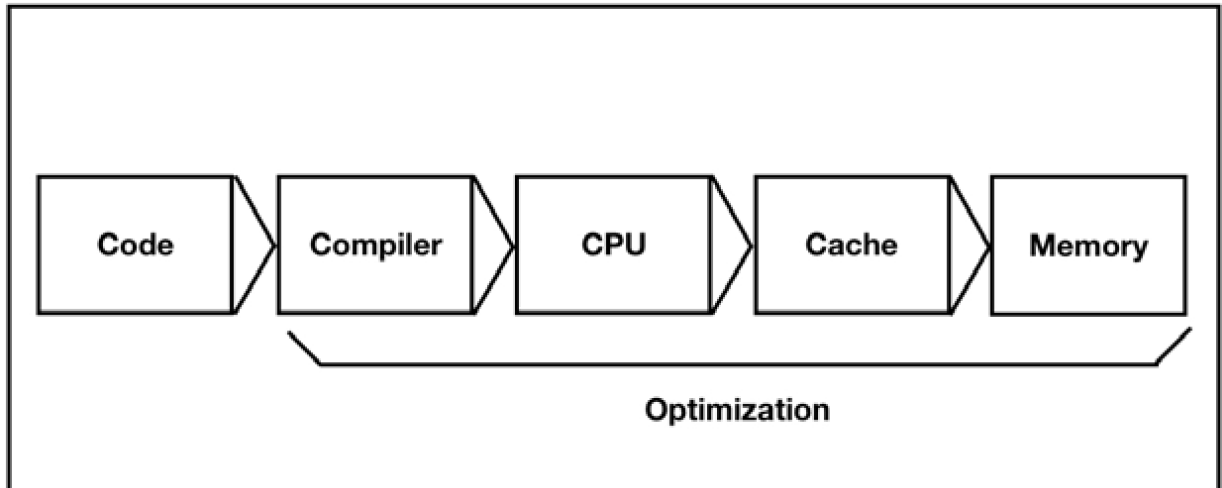
本章将介绍几种常见的线程同步方法，包括互斥锁、信号量、屏障、条件变量、读写锁和自旋锁。条件变量（Condition Variable）通常与互斥锁一起使用，用于线程间的等待和唤醒。信号量（Semaphores）可以用于控制对共享资源的访问数量，例如在有限数量的资源池中分配任务。屏障（Barrier）则用于确保所有线程都到达某个特定点后再继续执行。

本章将介绍几种常见的线程同步方法，包括互斥锁、信号量、屏障、条件变量、读写锁和自旋锁。互斥锁（Mutex）是最基本的同步机制，用于保证同一时刻只有一个线程可以访问共享资源。读写锁（RwLock）允许多个线程同时读取共享资源，但在写入时必须互斥。自旋锁（Spinlock）适用于处理器缓存一致性的系统，通过不断检查锁的状态来实现同步。

本章将介绍几种常见的线程同步方法，包括互斥锁、信号量、屏障、条件变量、读写锁和自旋锁。条件变量（Condition Variable）通常与互斥锁一起使用，用于线程间的等待和唤醒。信号量（Semaphores）可以用于控制对共享资源的访问数量，例如在有限数量的资源池中分配任务。屏障（Barrier）则用于确保所有线程都到达某个特定点后再继续执行。

本章将介绍几种常见的线程同步方法，包括互斥锁、信号量、屏障、条件变量、读写锁和自旋锁。

本章将介绍几种常见的线程同步方法，包括互斥锁、信号量、屏障、条件变量、读写锁和自旋锁。互斥锁（Mutex）是最基本的同步机制，用于保证同一时刻只有一个线程可以访问共享资源。读写锁（RwLock）允许多个线程同时读取共享资源，但在写入时必须互斥。自旋锁（Spinlock）适用于处理器缓存一致性的系统，通过不断检查锁的状态来实现同步。信号量（Semaphores）和屏障（Barrier）则用于更复杂的同步场景，如控制多个线程的执行顺序。



11-4 内存模型

- 内存模型描述了处理器如何看到内存
- 内存模型描述了处理器如何看到内存

内存模型描述了处理器如何看到内存。CPU 看到的内存模型是“顺序一致”的，即所有操作都是按顺序执行的。在 C++ 和 Rust 中，Atomic 类型用于保证内存模型的一致性。

Rust 和 C++11 都支持 Atomic 类型。store 和 load 操作是 CPU 对内存的读写操作。Memory Order 是 C++11 和 Rust 中用于指定内存顺序的枚举类型。Rust 和 LLVM 都支持 Memory Order。

内存模型

内存模型描述了处理器如何看到内存。内存模型是处理器对内存的抽象表示。内存模型是处理器对内存的抽象表示。

- 内存模型描述了处理器如何看到内存
- 内存模型描述了处理器如何看到内存
- 内存模型描述了处理器如何看到内存

内存模型描述了处理器如何看到内存。内存模型是处理器对内存的抽象表示。内存模型是处理器对内存的抽象表示。Rust 和 LLVM 都支持 Memory Order。

11.2 多线程

Rust 多线程使用 `std::thread` 模块

- 使用 `std::thread` 模块
- 使用 `std::sync` 模块中的 `Channel` 类型

11.2.1 多线程

Rust 多线程使用 `std::thread` 模块

11-7

```
1. use std::thread;
2. fn main() {
3.     let mut v = vec![];
4.     for id in 0..5 {
5.         let child = thread::spawn(move || {
6.             println!("in child: {}", id);
7.         });
8.         v.push(child);
9.     }
10.    println!("in main : join before ");
11.    for child in v {
12.        child.join();
13.    }
14.    println!("in main : join after");
15. }
```

11-7 使用 `std::thread::spawn` 方法

4-9 使用 `for` 循环

11-13 10-14 join main
 11-8
 11-8

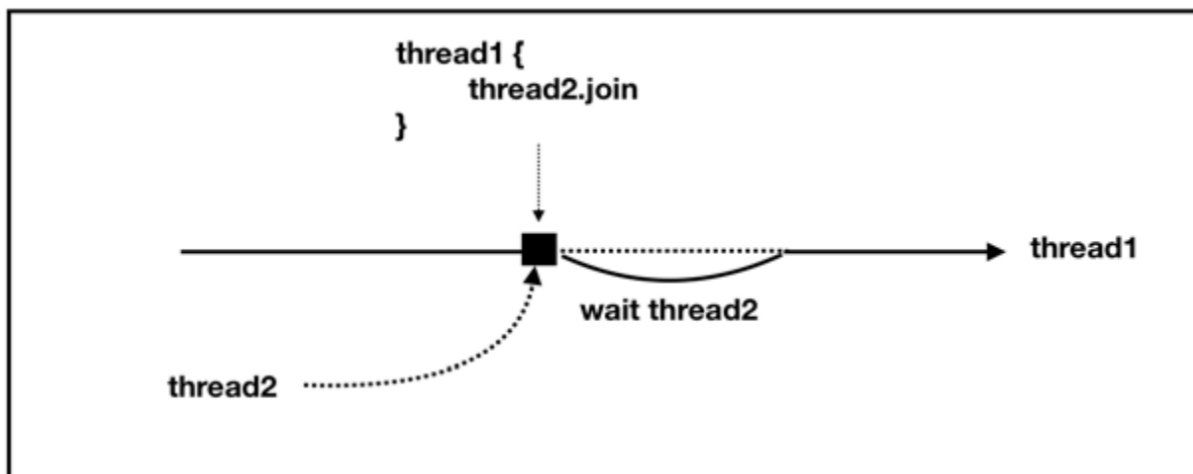
```

in child: 3
in child: 1
in child: 2
in child: 0
in main : join before
in child: 4
in main : join after
    
```

main “in main join before” “in main join after”

11-7 join main
 main
 main

join
 join 11-5



11-5 join

11-5 thread1 thread2 join thread1
 join thread2 thread2


```

    Builder { name:
stack_size:Rust
    RUST_MIN_STACK Builder{stack_size

```

```

    10builder.spawnspawnBuilder
threadspawnthreadspawn
Builder

```

```

    12 17 panic
catch_unwind catch_unwind
16 threadcurrent

```

```

    18spawnunwrapthread
spawnJoinHandleT Builderspawn
ResultJoinHandleTunwrapJoinHandleT
join

```

```

    11-911-10

```

```

    11-10

```

```

thread 'child-3' panicked at 'oh no!', src/main.rs:14:24
note: Run with `RUST_BACKTRACE=1` for a backtrace.

```

```

in child: 1

```

```

in child: 0

```

```

in child: 2

```

```

in child: 4

```

```

in child: 3

```

```

in child-3 do sm

```

```

    11-10"child-
3" "unknow"child-3

```

```


```

```

    Thread Local StorageTLS
11-11

```

```

    11-11

```

```

1. use std::cell::RefCell;
2. use std::thread;
3. fn main() {
4.     thread_local!(static FOO: RefCell<u32> = RefCell::new(1));
5.     FOO.with(|f| {
6.         assert_eq!(*f.borrow(), 1);
7.         *f.borrow_mut() = 2;
8.     });
9.     thread::spawn(|| {
10.        FOO.with(|f| {
11.            assert_eq!(*f.borrow(), 1);
12.            *f.borrow_mut() = 3;
13.        });
14.    });
15.    FOO.with(|f| {
16.        assert_eq!(*f.borrow(), 2);
17.    });
18. }

```

11-11 4 **thread_local** RefCell
 u32 1 FOO thread LocalKey
 FOO thread_local Cell RefCell

thread LocalKey with
 5 8 1
 borrow_mut 2

9 14 FOO 1
 thread_local LocalKey
 main FOO 1 12 FOO 3

15 17 FOO
 main FOO
 thread_local
 HashMap

11-11

std::thread park/unpark yield_now

std::thread park Thread::unpark park unpark park unpark std::thread park_timeout

11-12 park unpark

11-12 park unpark

```
1. use std::thread;
2. use std::time::Duration;
3. fn main() {
4.     let parked_thread = thread::Builder::new()
5.         .spawn(|| {
6.             println!("Parking thread");
7.             thread::park();
8.             println!("Thread unparked");
9.         }).unwrap();
10.    thread::sleep(Duration::from_millis(10));
11.    println!("Unpark the thread");
12.    parked_thread.thread().unpark();
13.    parked_thread.join().unwrap();
14. }
```

11-12 std::time::Duration new ns from_secs from_millis s ms

4 9 Builder thread park

10 thread sleep 10ms parked_thread sleep

12 parked_thread thread JoinHandle unpark parked_thread


```

1. pub fn spawn<F, T>(f: F) -> JoinHandle<T> where
2.     F: FnOnce() -> T, F: Send + 'static, T: Send + 'static
3. {
4.     Builder::new().spawn(f).unwrap()
5. }

```

11-14 spawn F T Send static Send Send Send Send Send

static T & static Rust

Arc T Arc T Rc T Rc T Arc T

11-15 Arc T Send Sync

11-15 Arc T Send Sync

```

1. unsafe impl<T: ?Sized + Sync + Send> Send for Arc<T> {}
2. unsafe impl<T: ?Sized + Sync + Send> Sync for Arc<T> {}

```

T Send Sync Arc T Send Sync Send Sync trait unsafe trait Rust std marker Send Sync

11-16

11-16 Send Sync


```

1. unsafe impl Send for .. { }
2. impl<T: ?Sized> !Send for *const T { }
3. impl<T: ?Sized> !Send for *mut T { }
4. unsafe impl Sync for .. { }
5. impl<T: ?Sized> !Sync for *const T { }
6. impl<T: ?Sized> !Sync for *mut T { }
7. mod impls {
8.     unsafe impl<'a, T: Sync + ?Sized> Send for &'a T {}
9.     unsafe impl<'a, T: Send + ?Sized> Send for &'a mut T {}
10. }

```

11-16 1 4 trait 1 4
 Send Sync Send Sync trait

- impl trait
- trait

2 3 5 6 *const T *mut T
 Send Sync trait

7 10 &a T &a mut T Send T
 &a T T Sync Sync
 &a mut T T Send Send

std marker Send Sync
 Cell RefCell Sync Rc Send

Send Sync 11-17

11-17

```

1. use std::thread;
2. fn main() {
3.     let mut s = "Hello".to_string();
4.     for _ in 0..3 {
5.         thread::spawn(move || {
6.             s.push_str(" Rust!");
7.         });
8.     }
9. }

```

11-17 s Rust Rust

error[E0382]: capture of moved value: `s`

```

5 |         thread::spawn(move || {
|             ----- value moved (into closure) here
6 |             s.push_str(" hello");
|             ^ value captured here after move

```

s Rust Rust

s Rc Arc Rc Send

11-18

11-18 Rc

```

1. use std::thread;
2. use std::rc::Rc;
3. fn main() {
4.     let mut s = Rc::new("Hello".to_string());
5.     for _ in 0..3 {
6.         let mut s_clone = s.clone();
7.         thread::spawn(move || {
8.             s_clone.push_str(" hello");
9.         });
10.    }
11. }

```

11-18 Rc s clone

```
error[E0277]: the trait bound `std::rc::Rc<std::string::String>:
  std::marker::Send` is not satisfied in `[closure@src/main.rs:
  7:23: 9:10 s_clone:std::rc::Rc<std::string::String>]`
7 |         thread::spawn(move || {
  |         ^^^^^^^^^^^^^^^ `std::rc::Rc<std::string::String>` cannot be
sent between threads safely
```

spawn Send Send Rc String Send Send Rc String Rc T

Rc T Arc T 11-19

11-19 Arc

```
1. use std::thread;
2. use std::sync::Arc;
3. fn main() {
4.     let s = Arc::new("Hello".to_string());
5.     for _ in 0..3 {
6.         let s_clone = s.clone();
7.         thread::spawn(move || {
8.             s_clone.push_str(" world!");
9.         });
10.    }
11. }
```

11-19 Arc Rc

```
error[E0596]: cannot borrow immutable borrowed content as mutable
8 |         s_clone.push_str(" world!");
  |         ^^^^^^^ cannot borrow as mutable
```

Arc T Cell RefCell Cell RefCell

11-20 Sync RefCell

11-20 RefCell

```
1. use std::thread;
2. use std::sync::Arc;
3. use std::cell::RefCell;
4. fn main() {
5.     let s = Arc::new(RefCell::new("Hello".to_string()));
6.     for _ in 0..3 {
7.         let s_clone = s.clone();
8.         thread::spawn(move || {
9.             let s_clone = s_clone.borrow_mut();
10.            s_clone.push_str(" world!");
11.        });
12.    }
13. }
```

11-20 RefCell

error[E0277]: the trait bound

`std::cell::RefCell<std::string::String>: std::marker::Sync` is not satisfied

```
8 |         thread::spawn(move || {
  |             ^^^^^^^^^^^^^^^ `std::cell::RefCell<std::string::String>`
cannot be shared between threads safely
```

RefCell String Sync Arc Sync

11.2.3

11-20 Rust Mutex T 11-21

11-21 Mutex

```

1. use std::thread;
2. use std::sync::{Arc, Mutex};
3. fn main() {
4.     let s = Arc::new(Mutex::new("Hello".to_string()));
5.     let mut v = vec![];
6.     for _ in 0..3 {
7.         let s_clone = s.clone();
8.         let child = thread::spawn(move || {
9.             let mut s_clone = s_clone.lock().unwrap();
10.            s_clone.push_str(" world!");
11.        });
12.        v.push(child);
13.    }
14.    for child in v {
15.        child.join().unwrap();
16.    }
17. }

```

11-21 Mutex
 push_str

Mutex

Mutex **T** Rust **T** Send
 Mutex **T** Send Sync
 lock

Mutex **T** lock LockResult MutexGuard
 T LockResult T std sync MutexGuard
RAII Mutex **T**
 try_lock MutexGuard **T**
 Err

Rust
 std thread JoinHandle
 join Result **T** Err

Err::unwrap::Err::
Err::

“**Posion**”11-22

11-22 “ ”

```
1. use std::sync::{Arc, Mutex};
2. use std::thread;
3. fn main() {
4.     let mutex = Arc::new(Mutex::new(1));
5.     let c_mutex = mutex.clone();
6.     let _ = thread::spawn(move || {
7.         let mut data = c_mutex.lock().unwrap();
8.         *data = 2;
9.         panic!("oh no");
10.    }).join();
11.    assert_eq!(mutex.is_poisoned(), true);
12.    match mutex.lock() {
13.        Ok(_) => unreachable!(),
14.        Err(p_err) => {
15.            let data = p_err.get_ref();
16.            println!("recovered: {}", data);
17.        }
18.    };
19. }
```

11-22610panic
8 “*” data data
MutexGuardTDerefDerefMut

11**is_poisoned** 12
18 main lock lock
ErrErrErrPoisonErrorT
get_refget_mutT16data
“recovered3”


```
rand=random
1115ifRustbool
rand=randomtruecontinue_positiv1
18total_flipslock19
iter_countstotal_flips
main811-24
11-24main
```



```

1. extern crate rand;
2. use std::thread;
3. use std::sync::{Arc, Mutex};
4. fn main() {
5.     let total_flips = Arc::new(Mutex::new(0));
6.     let completed = Arc::new(Mutex::new(0));
7.     let runs = 8;
8.     let target_flips = 10;
9.     for _ in 0..runs {
10.        let total_flips = total_flips.clone();
11.        let completed = completed.clone();
12.        thread::spawn(move || {
13.            flip_simulate(target_flips, total_flips);
14.            let mut completed = completed.lock().unwrap();
15.            *completed += 1;
16.        });
17.    }
18.    loop {
19.        let completed = completed.lock().unwrap();
20.        if *completed == runs {
21.            let total_flips = total_flips.lock().unwrap();
22.            println!("Final average: {}", *total_flips / *completed);
23.            break;
24.        }
25.    }
26. }
27. fn flip_simulate(target_flips: u64, total_flips: Arc<Mutex<u64>>) {
28.     // 同代码清单 11-23
29. }

```

□□□□□ 11-24 □□□□□□□□□□□□ total_flips □ completed□□□
total_flips□□□□□□□□□□□□completed□□□□□□□□□□□□□□□□

□□□9□17□□□□for□□□□8□□□□□□□□□□flip_simulate□□□□□□□□
□□□14□□□15□□□□□□□□□□completed□lock□□□□□□□□□□□□□□□□□□
□□□□□□□□□□

18-25 loop 20
11-24 11-25

11-25

11-25

```
iter_counts: 204
iter_counts: 1522
iter_counts: 1464
iter_counts: 1460
iter_counts: 1974
iter_counts: 423
iter_counts: 9913
iter_counts: 5447
Final average: 2800
```

Mutex T
Deadlock 11-26

11-26

11-26

```
1. fn main() {
2.     // 同代码清单 11-24
3.     // loop 循环的整段代码用下面四行代码来替换
4.     let completed = completed.lock().unwrap();
5.     while *completed < runs {}
6.     let total_flips = total_flips.lock().unwrap();
7.     println!("Final average: {}", *total_flips / *completed);
8. }
9. fn simulate(target_flips: u64, total_flips: Arc<Mutex<u64>>) {
10.    // 同代码清单 11-23
11. }
```

11-26 11-24 loop 4 7 loop break

Playgroud [2]

```
/root/entrypoint.sh: line 7:      5 Killed
timeout --signal=KILL ${timeout} "$@"
```

Playgroundmain
completed
completedmain

Rust

RwLock

std::sync——RwLockT
TMutexTRwLockTReader
WriterMutexTRwLockT
MutexTRwLockT“”

11-27

11-27

```
1. use std::sync::RwLock;
2. fn main() {
3.     let lock = RwLock::new(5);
4.     {
5.         let r1 = lock.read().unwrap();
6.         let r2 = lock.read().unwrap();
7.         assert_eq!(*r1, 5);
8.         assert_eq!(*r2, 5);
9.     }
10.    {
11.        let mut w = lock.write().unwrap();
12.        *w += 1;
13.        assert_eq!(*w, 6);
14.    }
15. }
```

11-27readwrite

□□□□□□ □

11.2.4 □□□□□□

Rust□□□□□□□□□□□□□□□□**Barrier**□ □□□□□**Condition**
Variable□ □□□□□□□□11-28□□□□□□□□□

□□□□**11-28**□□□□□

```
1. use std::sync::{Arc, Barrier};
2. use std::thread;
3. fn main() {
4.     let mut handles = Vec::with_capacity(5);
5.     let barrier = Arc::new(Barrier::new(5));
6.     for _ in 0..5 {
7.         let c = barrier.clone();
8.         handles.push(thread::spawn(move || {
9.             println!("before wait");
10.            c.wait();
11.            println!("after wait");
12.        }));
13.    }
14.    for handle in handles {
15.        handle.join().unwrap();
16.    }
17. }
```

□□□□□□□□□□□□□□□□**wait**□□□□□□□□□□□□□□□□□□□□□□□□10□□
□□□□□□□□□□□□□□□□11-29□□□

□□□□**11-29**□□□□□□□□□

before wait
before wait
before wait
before wait
before wait
after wait
after wait
after wait
after wait
after wait

5个线程同时等待“资源”资源等待wait5
个线程同时等待资源资源资源资源资源资源资源资源资源资源

资源 资源资源资源资源资源资源资源资源资源资源资源资源资源
资源11-30资源资源资源资源

资源11-30资源资源

```
1. use std::sync::{Arc, Condvar, Mutex};
2. use std::thread;
3. fn main() {
4.     let pair = Arc::new((Mutex::new(false), Condvar::new()));
5.     let pair_clone = pair.clone();
6.     thread::spawn(move || {
7.         let &(ref lock, ref cvar) = &*pair_clone;
8.         let mut started = lock.lock().unwrap();
9.         *started = true;
10.        cvar.notify_one();
11.    });
12.    let &(ref lock, ref cvar) = &*pair;
13.    let mut started = lock.lock().unwrap();
14.    while !*started {
15.        println!("{}", started); // false
16.        started = cvar.wait(started).unwrap();
17.        println!("{}", started); // true
18.    }
19. }
```

11-30 4 ArcMutexbool
Condvarpair

611locklock
bool true notify_one

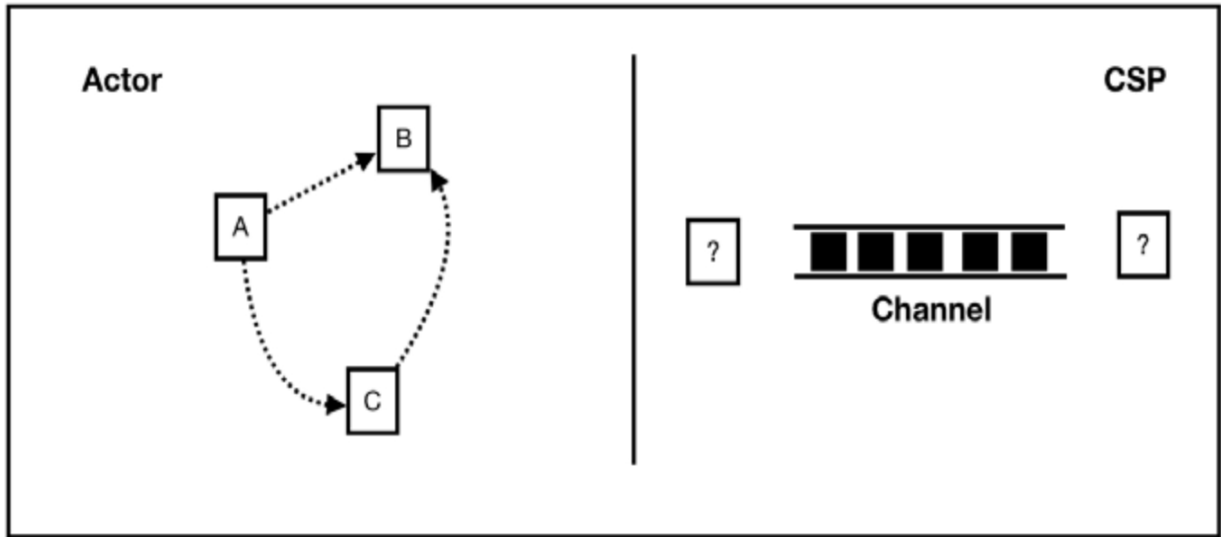
1218lockwhilewait
mainstartedtrue

11.2.5

“” Lock-Free Rust C++11

“” “”

- **Load**
- **Store**
- “-”
- **CASCompare-And-Swap**
- **Swap**
- **Compare-Exchange**



11-6 Actor 与 CSP 的区别

Actor 模型和 CSP 模型是两种不同的并发模型。Golang 和 Rust 都支持 Actor 模型，而 CSP 模型则主要用于理论研究和一些特定的并发库。

CSP 模型

CSP (Communicating Sequential Processes) 是一种并发模型，它强调进程之间的通信。在 CSP 模型中，进程通过通道 (Channel) 进行通信，通道可以看作是一个队列。CSP 模型通常用于描述和分析并发系统的行为。

CSP 模型与 Actor 模型的主要区别在于通信的方式。在 Actor 模型中，进程通过消息传递 (Message Passing) 进行通信，而消息是由 Actor 直接发送和接收的。而在 CSP 模型中，进程通过通道进行通信，通道起到了缓冲和排队的作用。

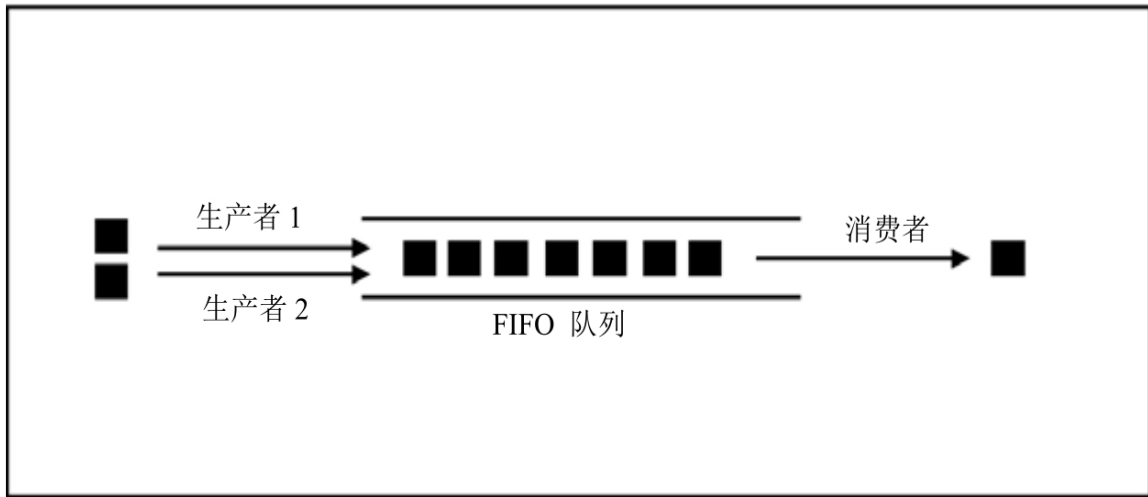
CSP 模型在理论研究和一些特定的并发库中得到了广泛应用。例如，在 Rust 中，std::sync::mpsc 就是一个基于 CSP 模型实现的并发库，它提供了 Multi-Producer-Single-Consumer (MPSC) 和 FIFO 类型的通道。而在 Golang 中，Channel 则是基于 Actor 模型实现的。

Channel 模型

Channel 模型是一种并发模型，它强调进程之间的通信。在 Channel 模型中，进程通过通道 (Channel) 进行通信，通道可以看作是一个队列。Channel 模型通常用于描述和分析并发系统的行为。

Channel 模型与 Actor 模型的主要区别在于通信的方式。在 Actor 模型中，进程通过消息传递 (Message Passing) 进行通信，而消息是由 Actor 直接发送和接收的。而在 Channel 模型中，进程通过通道进行通信，通道起到了缓冲和排队的作用。

Rust 中的 Channel 模型主要指的是 std::sync::mpsc 和 std::sync::mpsc::Sender 等类型。11-7



11-7 生产者-消费者问题

在 Rust 中，FIFO 队列可以通过 `CSP` 中的 `Channel` 实现。标准库 `std::sync::mpsc` 提供了基本的 `CSP` 实现。

- **Sender** 用于发送数据
- **SyncSender** 用于同步发送
- **Receiver** 用于接收数据

Rust 中的 `Channel` 实现如下：

- `Channel` 包含 `channel`、`Sender` 和 `Receiver`。


```

      struct Channel {
          channel: Vec<T>,
          sender: Sender,
          receiver: Receiver,
      }
      
```
- `Channel` 实现 `sync_channel` 方法，返回 `SyncSender` 和 `Receiver`。


```

      impl Channel {
          pub fn sync_channel(capacity: usize) -> (SyncSender, Receiver) {
              let (sender, receiver) = mpsc::sync_channel(capacity);
              let channel = Vec::new();
              (Channel { channel, sender, receiver }, channel)
          }
      }
      
```

`Channel` 实现 `Result` 类型，用于处理错误。如果发生错误，`unwrap` 方法会抛出异常。

在 11-33 中，我们使用 `Channel` 实现生产者-消费者问题。

在 11-33 中，我们使用 `Channel` 实现生产者-消费者问题。

```

1. use std::thread;
2. use std::sync::mpsc::channel;
3. fn main() {
4.     let (tx, rx) = channel();
5.     thread::spawn(move || {
6.         tx.send(10).unwrap();
7.     });
8.     assert_eq!(rx.recv().unwrap(), 10);
9. }

```

11-33 4 channel 3
 tx rx [3] Port —— 0 0

5 7 spawn tx send
 Channel

8 main rx recv
 Channel

11-33 Channel **Streaming Channel**
 Rust SPSC

11-34 Channel

11-34 Channel

```

1. use std::thread;
2. use std::sync::mpsc::channel;
3. fn main() {
4.     let (tx, rx) = channel();
5.     for i in 0..10 {
6.         let tx = tx.clone();
7.         thread::spawn(move || {
8.             tx.send(i).unwrap();
9.         });
10.    }
11.    for _ in 0..10 {
12.        let j = rx.recv().unwrap();
13.        assert!(0 <= j && j < 10);
14.    }
15. }

```

11-34 5 10 for 10 tx
10 10

11 14 for rx 10

11-34 Channel Sharing
Channel

Channel 11-35 Channel

11-35 Channel

```

1. use std::sync::mpsc::sync_channel;
2. use std::thread;
3. fn main() {
4.     let (tx, rx) = sync_channel(1);
5.     tx.send(1).unwrap();
6.     thread::spawn(move || {
7.         tx.send(2).unwrap();
8.     });
9.     assert_eq!(rx.recv().unwrap(), 1);
10.    assert_eq!(rx.recv().unwrap(), 2);
11. }

```

11-35 4 sync_channel
Channel 1

5 tx send Channel

6 8 spawn tx
Channel 1
Channel

9 10 rx Channel rx

Channel

11-36

11-36 Channel

```
1. use std::thread;
2. use std::sync::mpsc::channel;
3. fn main() {
4.     let (tx, rx) = channel();
5.     for i in 0..5 {
6.         let tx = tx.clone();
7.         thread::spawn(move || {
8.             tx.send(i).unwrap();
9.         });
10.    }
11.    // drop(tx);
12.    for j in rx.iter() {
13.        println!("{:?}", j);
14.    }
15. }
```

11-36 rx iter

```

/root/entrypoint.sh: line 7:      5 Killed
timeout --signal=KILL ${timeout} "$@"
0
1
4
2
3
4

```

Playground04
 entrypoint.sh11-36

main04main
 rxiter tx tx
 None main tx
 tx
 drop tx11-3611

11-37

11-37Channel

```

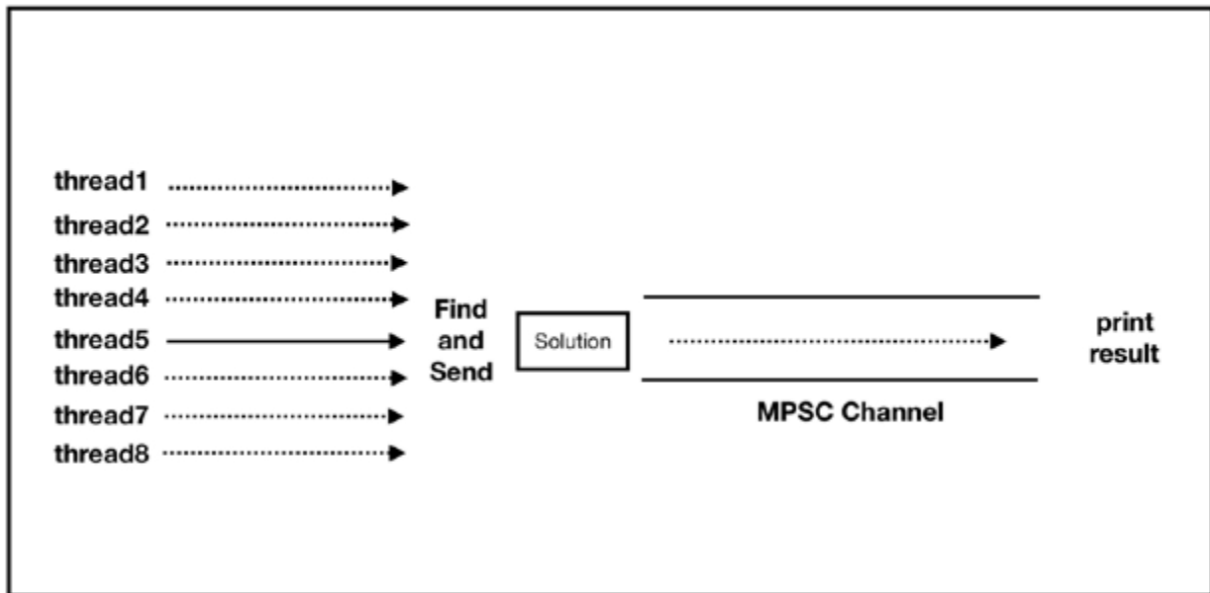
1. use std::sync::mpsc::channel;
2. use std::thread;

3. fn main() {
4.     let (tx, rx) = channel();
5.     thread::spawn(move || {
6.         tx.send(1u8).unwrap();
7.         tx.send(2u8).unwrap();
8.         tx.send(3u8).unwrap();
9.     });
10.    for x in rx.iter() {
11.        println!("receive: {}", x);
12.    }
13. }

```

11-37rxiter tx
 11-36

- 通过Channel实现多线程之间的通信
- 11-8



11-8

通过Cargo new--bin pow 创建一个新的二进制可执行文件，并添加依赖项rust-crypto和itertools。Cargo.toml文件如下所示：

11-38 Cargo.toml

1. [dependencies]
2. itertools = "0.7.8"
3. rust-crypto = "^0.2"

main.rs

11-39 main.rs

```

1. // extern crate itertools;
2. // extern crate crypto;
3. use itertools::Itertools;
4. use crypto::digest::Digest;
5. use crypto::sha2::Sha256;
6. use std::thread;
7. use std::sync::{mpsc, Arc};
8. use std::sync::atomic::{AtomicBool, Ordering};
9. const BASE: usize = 42;
10. const THREADS: usize = 8;
11. static DIFFICULTY: &'static str = "00000";
12. struct Solution(usize, String);

```

11-39 rust-crypto itertools
 Sha256 Rust 2018 1 2

6 8 MPSC Arc AtomicBool

9 10 BASE THREADS
 42

11 DIFFICULTY "0"
 "0"

12 Solution usize String

11-40

11-40 main.rs üerify

```

1. // 接上
2. fn verify(number: usize) -> Option<Solution> {
3.     let mut hasher = Sha256::new();
4.     hasher.input_str(&(number * BASE).to_string());
5.     let hash: String = hasher.result_str();
6.     if hash.starts_with(DIFFICULTY) {
7.         Some(Solution(number, hash))
8.     } else { None }
9. }

```

11-40 验证一个数字是否是 solution
Option<Solution>

3 5 使用 rust-crypto 中的 Sha256 对数字进行哈希
BASE 哈希结果 String

6 8 如果哈希结果以 DIFFICULTY 开头，则返回 Some(Solution(number, hash))，否则返回 None

11-41 线程池

11-41 main.rs 中的 find 函数

```

1. // 接上
2. fn find(
3.     start_at: usize,
4.     sender: mpsc::Sender<Solution>,
5.     is_solution_found: Arc<AtomicBool>
6. ) {
7.     for number in (start_at..).step(THREADS) {
8.         if is_solution_found.load(Ordering::Relaxed) { return; }
9.         if let Some(solution) = verify(number) {
10.             is_solution_found.store(true, Ordering::Relaxed);
11.             sender.send(solution).unwrap();
12.             return;
13.         }
14.     }
15. }

```

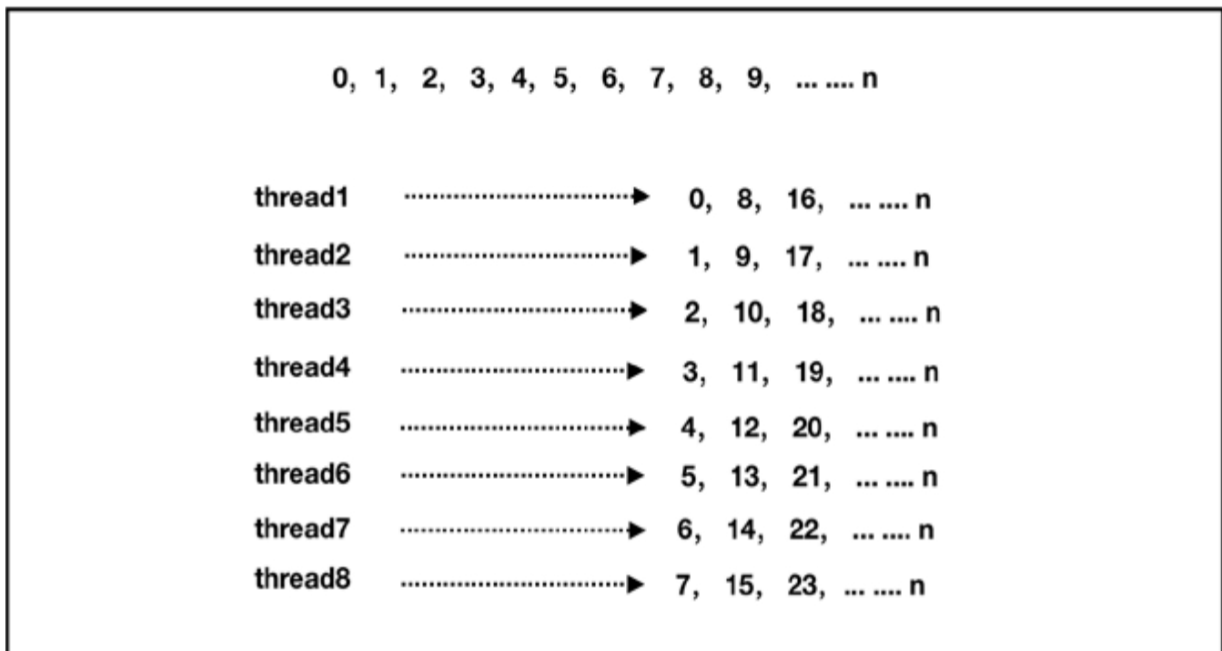
11-41 find 0 1 2 3 4 5 6 7 8 9 ... n
 sender Channel mpsc Sender Solution
 Solution Channel
is_solution_found Solution
 Arc AtomicBool

7 start_at THREADS
 THREADS

8 load is_solution_found
 true Ordering
Relaxed find

9 13 verify number
 store is_solution_found true
 solution Channel

11-9



11-9

11-9 8 8

mian 11-42

11-42 main.rs main

```
1. // 接上
2. fn main() {
3.     println!("PoW : Find a number,
4.         SHA256(the number * {}) == \"{}.....\" ", BASE, DIFFICULTY);
5.     println!("Started {} threads", THREADS);
6.     println!("Please wait... ");
7.     let is_solution_found = Arc::new(AtomicBool::new(false));
8.     let (sender, receiver) = mpsc::channel();
9.     for i in 0..THREADS {
10.        let sender_n = sender.clone();
11.        let is_solution_found = is_solution_found.clone();
12.        thread::spawn(move || {
13.            find(i, sender_n, is_solution_found);
14.        });
15.    }
16.    match receiver.recv() {
17.        Ok(Solution(i, hash)) => {
18.            println!("Found the solution: ");
19.            println!("The number is: {},
20.                and hash result is : {}.\"", i, hash);
21.        },
22.        Err(_) => panic!("Worker threads disconnected!"),
23.    }
24. }
```

11-42 36

7 8 is_solution_found Channel
is_solution_found false

915THREADSfind
find

1623receiverrecvmain

cargo run
11-43

11-43

PoW : Find a number, SHA256(the number * 42) == "00000....."

Started 8 threads

Please wait...

Found the solution:

The number is: 834312, and hash result is :
00000a31988d8c179097b2753c509b11520f4b5470dc77facedc5734f13d3394.

-
-
-

11.2.7

RustCellTRefCellT
11-44Mutex

11-44Mutex

```
1. pub struct Mutex<T: ?Sized> {  
2.     inner: Box<sys::Mutex>,  
3.     poison: poison::Flag,  
4.     data: UnsafeCell<T>,  
5. }
```

11-44MutexTinnerpoison
datainner API sysMutexpoison
“”data UnsafeCellT
UnsafeCellT

Cell<T> RefCell<T> RwLock<T> mpsc<T>
Sender 11-45

11-45 Cell<T> RefCell<T> RwLock<T>

```
1. pub struct Cell<T> {
2.     value: UnsafeCell<T>,
3. }
4. pub struct RefCell<T: ?Sized> {
5.     borrow: Cell<BorrowFlag>,
6.     value: UnsafeCell<T>,
7. }
8. pub struct RwLock<T: ?Sized> {
9.     inner: Box<sys::RWLock>,
10.    poison: poison::Flag,
11.    data: UnsafeCell<T>,
12. }
13. pub struct AtomicBool {
14.     v: UnsafeCell<u8>,
15. }
16. pub struct Sender<T> {
17.     inner: UnsafeCell<Flavor<T>>,
18. }
19. pub struct Receiver<T> {
20.     inner: UnsafeCell<Flavor<T>>,
21. }
```

11-45 UnsafeCell<T>
UnsafeCell<T> 11-46

11-46 UnsafeCell<T>

```

1. #[lang = "unsafe_cell"]
2. pub struct UnsafeCell<T: ?Sized> {
3.     value: T,
4. }
5. impl<T: ?Sized> !Sync for UnsafeCell<T> {}
6. impl<T: ?Sized> UnsafeCell<T> {
7.     pub fn get(&self) -> *mut T {
8.         &self.value as *const T as *mut T
9.     }
10. }

```

11-46 UnsafeCell<T> Lang Item

UnsafeCell<T> Sync 7 8 get Raw Pointer get as T *counst T *mut T

Rust UnsafeCell<T> UnsafeCell<T> Rust UnsafeCell<T>

UnsafeCell<T> Rust

11.2.8

-
-
-

• 在 Rust 中，线程池的实现通常依赖于 `mpsc` 通道和 `Builder` 来创建和管理线程。

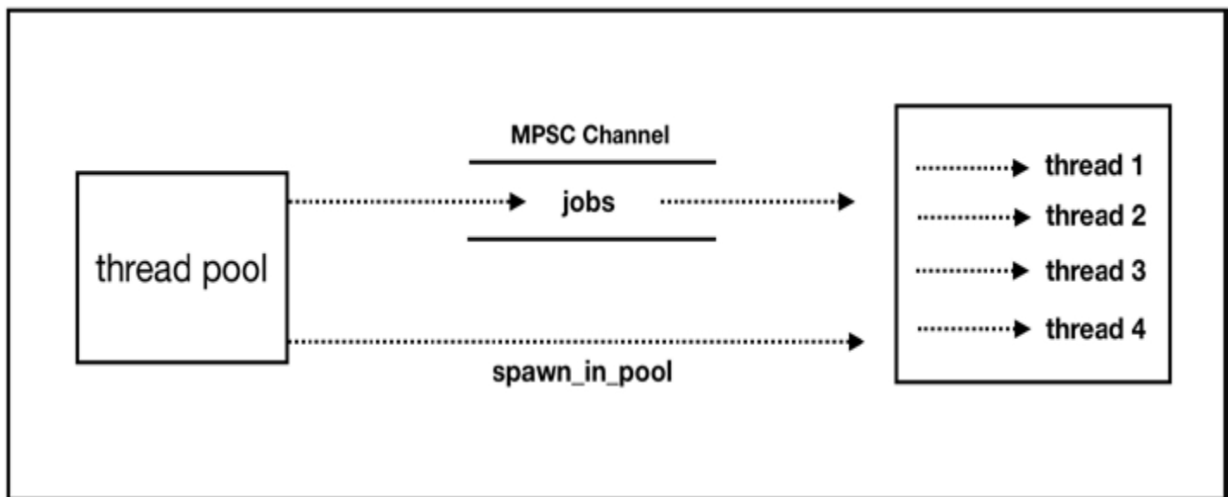
在 Rust 中，`threadpool` [4] 是一个常用的线程池库，它提供了简单易用的接口来创建和管理线程池。

• 在 Rust 中，线程池的实现通常依赖于 `mpsc` 通道和 `Builder` 来创建和管理线程。

• 在 Rust 中，线程池的实现通常依赖于 `mpsc` 通道和 `Builder` 来创建和管理线程。

• 在 Rust 中，线程池的实现通常依赖于 `mpsc` 通道和 `Builder` 来创建和管理线程。

在 Rust 中，线程池的实现通常依赖于 `mpsc` 通道和 `Builder` 来创建和管理线程。



在 Rust 中，线程池的实现通常依赖于 `mpsc` 通道和 `Builder` 来创建和管理线程。

在 Rust 中，线程池的实现通常依赖于 `mpsc` 通道和 `Builder` 来创建和管理线程。

在 Rust 中，线程池的实现通常依赖于 `mpsc` 通道和 `Builder` 来创建和管理线程。

1. `[dependencies]`
2. `num_cpus = "1.8"`

在 Rust 中，线程池的实现通常依赖于 `mpsc` 通道和 `Builder` 来创建和管理线程。

在 Rust 中，线程池的实现通常依赖于 `mpsc` 通道和 `Builder` 来创建和管理线程。

在 Rust 中，线程池的实现通常依赖于 `mpsc` 通道和 `Builder` 来创建和管理线程。

```

1. // extern crate num_cpus;
2. use std::sync::mpsc::{channel, Sender, Receiver};
3. use std::sync::{Arc, Mutex, Condvar};
4. use std::sync::atomic::{AtomicUsize, Ordering};
5. use std::thread;
6. trait FnBox {
7.     fn call_box(self: Box<Self>);
8. }
9. impl<F: FnOnce()> FnBox for F {
10.     fn call_box(self: Box<F>) {
11.         (*self)()
12.     }
13. }
14. type Thunk<'a> = Box<FnBox + Send + 'a>;

```

11-48 `num_cpus` 2 `channel`
`Sender` `Receiver` 3 `Arc`
`Mutex` `Condvar` `AtomicUsize` 5 `thread`

6 13 `FnBox` trait `FnOnce`
 trait `call_box` **[feature `fnbox`]**
 6

14 `type`
 11-49
 11-49

11-49 `main.rs` `ThreadPoolSharedData`

```

1. // 接上
2. struct ThreadPoolSharedData {
3.     name: Option<String>,
4.     job_receiver: Mutex<Receiver<Thunk<'static>>>,
5.     empty_trigger: Mutex<()>,
6.     empty_condvar: Condvar,
7.     queued_count: AtomicUsize,
8.     active_count: AtomicUsize,
9.     max_thread_count: AtomicUsize,
10.    panic_count: AtomicUsize,
11.    stack_size: Option<usize>,
12. }
13. impl ThreadPoolSharedData {
14.     fn has_work(&self) -> bool {
15.         self.queued_count.load(Ordering::SeqCst) > 0
16.         ||
17.         self.active_count.load(Ordering::SeqCst) > 0
18.     }
19.     fn no_work_notify_all(&self) {
20.         if !self.has_work() {
21.             *self.empty_trigger.lock()
22.                 .expect("Unable to notify all joining threads");
23.             self.empty_condvar.notify_all();
24.         }
25.     }
26. }

```

11-49 ThreadPoolSharedData 212

· **name** Option<String>

· **job_receiver** Channel rx
Mutex<Receiver<Thunk<'static>>>Reveiver

```
    Thunk::static::new().start().join().Thunk::
    static::Box::FnBox+Send+static::new().start().join().
```

```
    · empty_trigger & empty_condvar & Mutex & Condvar
    Condvar::new().start().join().
```

```
    · queued_count & active_count & AtomicUsize
    AtomicUsize::new().
```

```
    · max_thread_count & AtomicUsize
```

```
    · panic_count & AtomicUsize
    AtomicUsize::new().
```

```
    · stack_size & Option<usize>
    Option::new().8MB
```

```
    14-18 ThreadPoolSharedData::has_work
    queued_count=0 active_count=0
```

```
    19-25 no_work_notify_all has_work
    empty_trigger empty_condvar notify_all
    join
```

```
    11-50
```

```
    11-50 main.rs ThreadPool
```

```

1. // 接上
2. pub struct ThreadPool {
3.     jobs: Sender<Thunk<'static>>,
4.     shared_data: Arc<ThreadPoolSharedData>,
5. }
6. impl ThreadPool {
7.     pub fn new(num_threads: usize) -> ThreadPool {
8.         Builder::new().num_threads(num_threads).build()
9.     }
10.    pub fn execute<F>(&self, job: F)
11.        where F: FnOnce() + Send + 'static
12.    {
13.        self.shared_data
14.            .queued_count.fetch_add(1, Ordering::SeqCst);
15.        self.jobs.send(Box::new(job))
16.            .expect("unable to send job into queue.");
17.    }
18.    pub fn join(&self) {
19.        if self.shared_data.has_work() == false {
20.            return ();
21.        }
22.        let mut lock = self.shared_data.empty_trigger.lock().unwrap();
23.        while self.shared_data.has_work() {
24.            lock = self.shared_data
25.                .empty_condvar.wait(lock).unwrap();
26.        }
27.    }
28. }

```

11-50ThreadPool25

• **jobs** Channel tx Sender<Thunk<'static>>

· **shared_data** Arc
ThreadPoolSharedData

new

execute Channel
AtomicUsize::fetch_add(queued_count

join
19 21
shared_data.empty_trigger
23 26 while
empty_condvar::wait
notify_all

Builder::build 11-51

11-51 main.rs Builder

```

1. // 接上
2. #[derive(Clone, Default)]
3. pub struct Builder {
4.     num_threads: Option<usize>,
5.     thread_name: Option<String>,
6.     thread_stack_size: Option<usize>,
7. }
8. impl Builder {
9.     pub fn new() -> Builder {
10.         Builder {
11.             num_threads: None,
12.             thread_name: None,
13.             thread_stack_size: None,
14.         }
15.     }
16.     pub fn num_threads(mut self, num_threads: usize) -> Builder {
17.         assert!(num_threads > 0);
18.         self.num_threads = Some(num_threads);
19.         self
20.     }
21.     pub fn build(self) -> ThreadPool {
22.         let (tx, rx) = channel::<Thunk<'static>>();
23.         let num_threads = self.num_threads
24.             .unwrap_or_else(num_cpus::get);
25.         let shared_data = Arc::new(ThreadPoolSharedData {
26.             name: self.thread_name,
27.             job_receiver: Mutex::new(rx),
28.             empty_condvar: Condvar::new(),
29.             empty_trigger: Mutex::new(()),
30.             queued_count: AtomicUsize::new(0),
31.             active_count: AtomicUsize::new(0),
32.             max_thread_count: AtomicUsize::new(num_threads),
33.             panic_count: AtomicUsize::new(0),
34.             stack_size: self.thread_stack_size,
35.         });
36.         for _ in 0..num_threads {
37.             spawn_in_pool(shared_data.clone());
38.         }
39.         ThreadPool {
40.             jobs: tx,
41.             shared_data: shared_data,
42.         }
43.     }
44. }

```

11-51 Builder num_threads
thread_name thread_stack_size
3 7

9 15 Builder new None
Builder

16 20 num_threads

21 43 build 22
channel 23 24 num_threads
num_cpus get CPU 25 35
ThreadPoolSharedData Arc 36 38
num_threads spawn_in_pool
39 42 ThreadPool

11-52 spawn_in_pool

11-52 spawn_in_pool


```
1. // 接上
2. fn spawn_in_pool(shared_data: Arc<ThreadPoolSharedData>) {
3.     let mut builder = thread::Builder::new();
4.     if let Some(ref name) = shared_data.name {
5.         builder = builder.name(name.clone());
6.     }
7.     if let Some(ref stack_size) = shared_data.stack_size {
8.         builder = builder.stack_size(stack_size.to_owned());
9.     }
10.    builder.spawn(move || {
11.        let sentinel = Sentinel::new(&shared_data);
12.        loop {
13.            let thread_counter_val = shared_data
14.                .active_count.load(Ordering::Acquire);
15.            let max_thread_count_val = shared_data
16.                .max_thread_count.load(Ordering::Relaxed);
17.            if thread_counter_val >= max_thread_count_val {
18.                break;
19.            }
20.            let message = {
21.                let lock = shared_data.job_receiver.lock()
22.                    .expect("unable to lock job_receiver");
23.                lock.recv()
24.            };
25.            let job = match message {
26.                Ok(job) => job,
27.                Err(..) => break,
28.            };
29.            shared_data.queued_count.fetch_sub(1, Ordering::SeqCst);
30.            shared_data.active_count.fetch_add(1, Ordering::SeqCst);
31.            job.call_box();
32.            shared_data.active_count.fetch_sub(1, Ordering::SeqCst);
```

```

33.         shared_data.no_work_notify_all();
34.     }
35.     sentinel.cancel();
36. }) .unwrap();
37. }

```

11-52 3 9 shared_data name
 stack_size thread Builder new
 main.rs Builder

10 36 builder.spawn
 11 Sentinel
 12 34 loop
 13 14 active_count load
 Ordering Acquire load active_count
 15 16 max_thread_count
 Ordering Relaxed max_thread_count
 17 19
 20 24 job_receiver recv
 25 28 match message message

29 30 shared_data queued_count 1
 1 active_count fetch_add
 1

31 job call_box
 32 active_count 1
 33 shared_data no_work_notify_all
 wait

35 cancel sentinel
 11-53 Sentinel

33 shared_data no_work_notify_all
 wait
 35 cancel sentinel
 11-53 Sentinel

例11-53 Sentinel

```
1. //接上
2. struct Sentinel<'a> {
3.     shared_data: &'a Arc<ThreadPoolSharedData>,
4.     active: bool,
5. }
6. impl<'a> Sentinel<'a> {
7.     fn new(shared_data: &'a Arc<ThreadPoolSharedData>)
8.         -> Sentinel<'a> {
9.         Sentinel {
10.             shared_data: shared_data,
11.             active: true,
12.         }
13.     }
14.     fn cancel(mut self) {
15.         self.active = false;
16.     }
17. }
18. impl<'a> Drop for Sentinel<'a> {
19.     fn drop(&mut self) {
20.         if self.active {
21.             self.shared_data.active_count
22.                 .fetch_sub(1, Ordering::SeqCst);
23.             if thread::panicking() {
24.                 self.shared_data.panic_count
25.                     .fetch_add(1, Ordering::SeqCst);
26.             }
27.             self.shared_data.no_work_notify_all();
28.             spawn_in_pool(self.shared_data.clone())
29.         }
30.     }
31. }
```

例 11-53 2.5 版 Sentinel 结构体，包含 shared_data 和 active 两个成员，以及 &'a Arc<ThreadPoolSharedData>

```
bool shared_data
active true false

```

```
6 17 new cancel Sentinel a
active false

```

```
18 30 Sentinel a Drop
11-52 Sentinel a
drop Sentinel a true
21 28

```

```
21 22 active_count 1
23 26 thread panicking
panic_count 1 27 shared_data
no_work_notify_all wait
28 spawn_in_pool

```

```
main 11-54

```

```
11-54 main

```

```
1. fn main() {
2.     let pool = ThreadPool::new(8);
3.     let test_count = Arc::new(AtomicUsize::new(0));
4.     for _ in 0..42 {
5.         let test_count = test_count.clone();
6.         pool.execute(move || {
7.             test_count.fetch_add(1, Ordering::Relaxed);
8.         });
9.     }
10.    pool.join();
11.    assert_eq!(42, test_count.load(Ordering::Relaxed));
12. }
```

```
11-54 ThreadPool new 8 8
2

```

```
3 test_count

```


11-55 1 2 rayon prelude 3
5 sum_of_squares par_iter Rayon
11-56

6 8 increment_all par_iter_mut
Rayon

main sum_of_squares increment_all
11-56

11-56

385

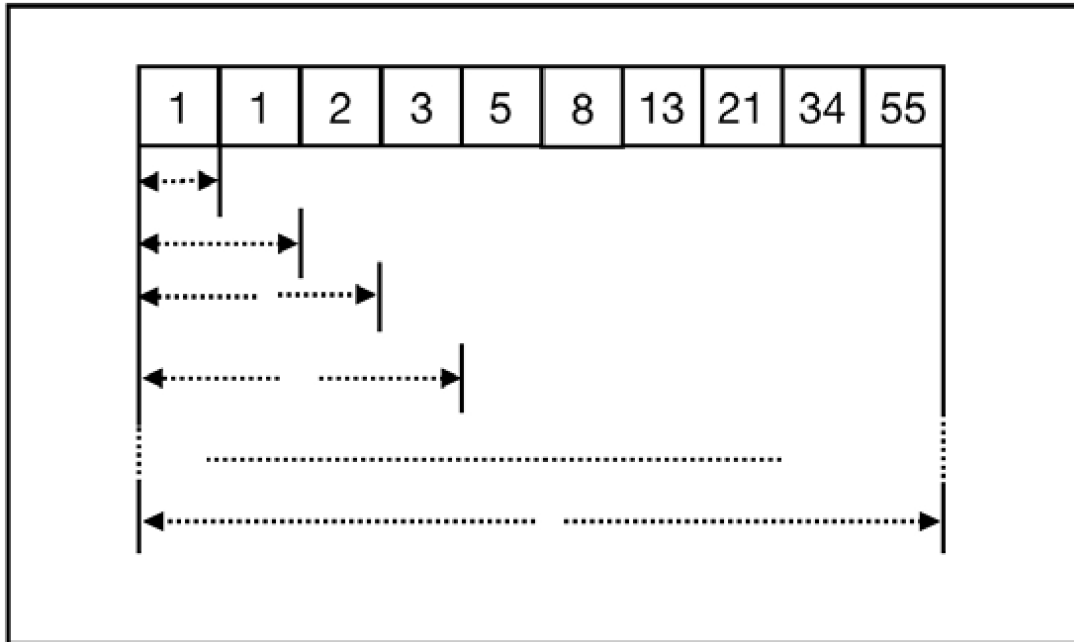
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

11-57 join

11-57 join

```
1. extern crate rayon;
2. fn fib(n: u32) -> u32 {
3.     if n < 2 { return n; }
4.     let (a, b) = rayon::join(
5.         || fib(n - 1), || fib(n - 2)
6.     );
7.     a + b
8. }
9. fn main() {
10.    let r = fib(32);
11.    assert_eq!(r, 2178309);
12. }
```

11-57 2 8 fib
rayon join 11-11



11-11 Rayon join 的實現

Rayon join 的實現是基於 Rayon 的 Work-Stealing 模型。在 Rayon 中，每個 worker 都有一個本地隊列，用於存放待處理的任務。當一個 worker 完成其本地隊列中的任務時，它會嘗試從其他 worker 的隊列中偷取任務。這種工作-stealing 模型使得 Rayon 能夠有效地利用多核處理器的資源，並實現並行計算。

Rayon 的實現基於 rayon-demo 庫。

11.2.10 Crossbeam

Crossbeam 是一個基於 Rust 語言的並行庫，它提供了一個簡單且高效的並行模型。Crossbeam 的實現基於 Rust 的 AtomicConsume 和 ArcCell 等原語。

- Crossbeam 基於 std::sync 庫實現，C++11 的 Consume 原語。
- Crossbeam 基於 thread 庫實現，C++11 的 Scoped 原語。
- Crossbeam 基於 CachePadded 庫實現。
- Crossbeam 基於 MPMC Channel 庫實現，C++11 的 MS-Queue 庫。

· crossbeam-epoch
Epoch GC
Hazard
Pointer
HP
QSBR
Quiescent-State-Based Reclamation

Crossbeam
crossbeam-utils
AtomicConsume
trait
"Consume"
"Consume"
C++
ARM
AArch64

crossbeam-utils
std
sync
atomic
AtomicBool
AtomicUsize
trait
load_consume

crossbeam-utils
AtomicCell
Cell
T

Scoped

11-58

11-58

```
1. fn main() {
2.     let array = [1, 2, 3];
3.     let mut guards = vec![];
4.     for &i in &array {
5.         let guard = std::thread::spawn(move || {
6.             println!("element: {}", i);
```



```

7.         });
8.         guards.push(guard);
9.     }
10.    for guard in guards {
11.        guard.join().unwrap();
12.    }
13. }

```

11-58 for &i

Crossbeam Scoped 11-59

11-59 Crossbeam Scoped

```

1. // extern crate crossbeam;
2. use crossbeam::thread::scope;
3. fn main() {
4.     let array = [1, 2, 3];
5.     scope(|scope| {
6.         for i in &array {
7.             scope.spawn(move || { println!("element: {}", i); });
8.         }
9.     });
10. }

```

11-59 1 Rust 2018 crossbeam thread scope scope main array

scope Scope join

False Sharing CPU Cache Line 64 L1

在 Rust 中，我们使用 `Crossbeam` 库来创建多线程的通道。这个库提供了多种类型的通道，包括有缓冲的通道、无缓冲的通道、带缓存的通道等。在本文中，我们将介绍如何使用 `Crossbeam` 库来创建多线程的通道。

在 Rust 中，我们使用 `Crossbeam` 库来创建多线程的通道。这个库提供了多种类型的通道，包括有缓冲的通道、无缓冲的通道、带缓存的通道等。在本文中，我们将介绍如何使用 `Crossbeam` 库来创建多线程的通道。

在 Rust 中，我们使用 `Crossbeam` 库来创建多线程的通道。这个库提供了多种类型的通道，包括有缓冲的通道、无缓冲的通道、带缓存的通道等。在本文中，我们将介绍如何使用 `Crossbeam` 库来创建多线程的通道。

MPMC Channel

`Crossbeam` 库提供了多种类型的通道，包括有缓冲的通道、无缓冲的通道、带缓存的通道等。在本文中，我们将介绍如何使用 `Crossbeam` 库来创建多线程的通道。

在 Rust 中，我们使用 `Crossbeam` 库来创建多线程的通道。这个库提供了多种类型的通道，包括有缓冲的通道、无缓冲的通道、带缓存的通道等。在本文中，我们将介绍如何使用 `Crossbeam` 库来创建多线程的通道。

11-60 Crossbeam MPMC Channel

```
1. use crossbeam::channel as channel;
2. fn main(){
3.     let (s, r) = channel::unbounded();
4.     crossbeam::scope(|scope| {
5.         scope.spawn(|| {
6.             s.send(1);
7.             r.recv().unwrap();
8.         });
9.         scope.spawn(|| {
10.            s.send(2);
11.            r.recv().unwrap();
12.        });
13.    });
```

在 Rust 2018 中，我们使用 `extern crate crossbeam;` 来引入 `Crossbeam` 库。在本文中，我们将介绍如何使用 `Crossbeam` 库来创建多线程的通道。


```

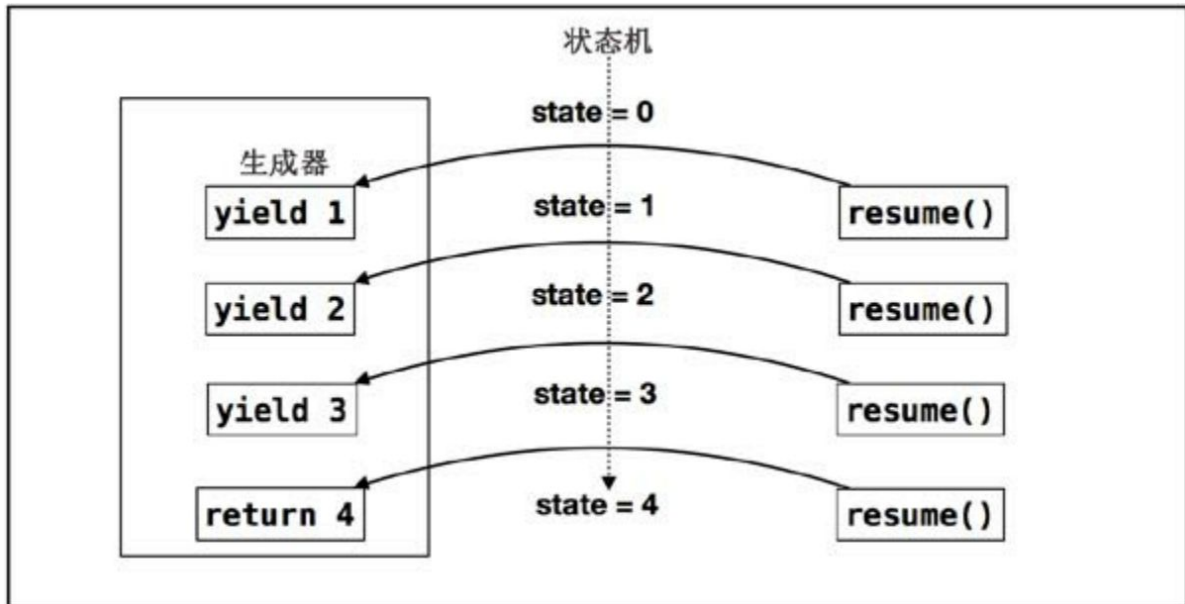
1.  #![feature(generators, generator_trait)]
2.  use std::ops::Generator;
3.  fn main(){
4.      let mut gen = ||{
5.          yield 1;
6.          yield 2;
7.          yield 3;
8.          return 4;
9.      };
10.     unsafe {
11.         for _ in 0..4{
12.             let c = gen.resume();
13.             println!("{:?}", c);
14.         }
15.     }
16. }

```

11-62 1 **feature generators generator_trait** Rust
 2 std ops Generator trait trait

4 9 Generator
yield

10 unsafe unsafe resume for
 4 resume yield
 11-12



11-12 生成器与状态机

11-12 生成器与状态机。生成器是一个特殊的函数，它返回一个生成器对象。生成器对象是一个迭代器，它可以在需要的时候生成下一个值。生成器对象内部维护了一个状态机，这个状态机记录了生成器当前的状态。当调用生成器对象的 `next()` 方法时，状态机会根据当前的状态来决定下一个要执行的代码块，并返回一个包含下一个值和下一个状态的元组。当生成器对象返回 `yield` 语句后面的那个值时，会将内部状态记下来，下一次再遇到 `yield` 语句时，就接着从上次那个位置开始执行。当遇到 `return` 语句时，就返回 `return` 语句后面的那个值，并将内部状态记下来，下一次再遇到 `yield` 语句时，就接着从上次那个位置开始执行。

生成器对象内部维护了一个状态机，这个状态机记录了生成器当前的状态。当调用生成器对象的 `next()` 方法时，状态机会根据当前的状态来决定下一个要执行的代码块，并返回一个包含下一个值和下一个状态的元组。当生成器对象返回 `yield` 语句后面的那个值时，会将内部状态记下来，下一次再遇到 `yield` 语句时，就接着从上次那个位置开始执行。当遇到 `return` 语句时，就返回 `return` 语句后面的那个值，并将内部状态记下来，下一次再遇到 `yield` 语句时，就接着从上次那个位置开始执行。

11-63 生成器与状态机

```
Yielded(1)
Yielded(2)
Yielded(3)
Complete(4)
```

11-12 生成器与状态机。生成器是一个特殊的函数，它返回一个生成器对象。生成器对象是一个迭代器，它可以在需要的时候生成下一个值。生成器对象内部维护了一个状态机，这个状态机记录了生成器当前的状态。当调用生成器对象的 `next()` 方法时，状态机会根据当前的状态来决定下一个要执行的代码块，并返回一个包含下一个值和下一个状态的元组。当生成器对象返回 `yield` 语句后面的那个值时，会将内部状态记下来，下一次再遇到 `yield` 语句时，就接着从上次那个位置开始执行。当遇到 `return` 语句时，就返回 `return` 语句后面的那个值，并将内部状态记下来，下一次再遇到 `yield` 语句时，就接着从上次那个位置开始执行。

生成器对象

Rust 的 `Generator` trait 11-64

11-64 Generator trait

```
1. pub trait Generator {  
2.     type Yield;  
3.     type Return;  
4.     unsafe fn resume(&mut self) ->  
5.         GeneratorState<Self::Yield, Self::Return>;  
6. }
```

11-64 Generator Yield Return
yield

11-62 gen
Generator 11-65

11-65 11-62 gen


```

1.  #![feature(generators, generator_trait)]
2.  use std::ops::{Generator, GeneratorState};
3.  enum __Gen {
4.      Start,
5.      State1(State1),
6.      State2(State2),
7.      State3(State3),
8.      Done
9.  }
10. struct State1 { x: u64 }
11. struct State2 { x: u64 }
12. struct State3 { x: u64 }
13. impl Generator for __Gen {
14.     type Yield = u64;
15.     type Return = u64;
16.     unsafe fn resume(&mut self) -> GeneratorState<u64, u64> {
17.         match std::mem::replace(self, __Gen::Done) {
18.             __Gen::Start=> {
19.                 *self = __Gen::State1(State1{x: 1});
20.                 GeneratorState::Yielded(1)
21.             }
22.             __Gen::State1(State1{x: 1}) => {
23.                 *self = __Gen::State2(State2{x: 2});
24.                 GeneratorState::Yielded(2)
25.             }
26.             __Gen::State2(State2{x: 2}) => {
27.                 *self = __Gen::State3(State3{x: 3});
28.                 GeneratorState::Yielded(3)
29.             }
30.             __Gen::State3(State3{x: 3}) => {
31.                 *self = __Gen::Done;
32.                 GeneratorState::Complete(4)
33.             }
34.             _ => {
35.                 panic!("generator resumed after completion")
36.             }
37.         }
38.     }
39. }
40. fn main() {
41.     let mut gen = __Gen::Start;
42.     for _ in 0..4 {
43.         println!("{:?}", unsafe{ gen.resume() });
44.     }
45. }

```

11-65 [feature generators generator_trait] Generator Nightly

__Gen 11-62 gen yield return 4 __Gen 4 State1 State1 State2 State2 State1 State3 Done Start 3 9

Start Done State1 State2 State3 10 12

13 __Gen Generator 14 15 Yield Return u64

16 unsafe resume resume std mem replace &mut self __Gen Done replace self __Gen Done self match replace

18 match __Gen Start replace __Gen Start State1 self __Gen State1 State1 {x 1} GeneratorState Yielded 1 resume self

main gen __Gen Start 4 4 resume 11-63

11-65

Return Yield Generator Yield=T Return= 11-66

11-66

```

1.  #![feature(generators, generator_trait)]
2.  use std::ops::{Generator, GeneratorState};
3.  pub fn up_to() -> impl Generator<Yield = u64, Return = ()> {
4.      || {
5.          let mut x = 0;
6.          loop {
7.              x += 1;
8.              yield x;
9.          }
10.         return ();
11.     }
12. }
13. fn main(){
14.     let mut gen = up_to();
15.     unsafe {
16.         for _ in 0..10{
17.             match gen.resume() {
18.                 GeneratorState::Yielded(i) => println!("{:?}", i),
19.                 _ => println!("Completed"),
20.             }
21.         }
22.     }
23. }

```

11-66 up_to impl Generator Yield=64
 Return= Rust 2018 Nightly Rust
impl Trait Rust 2018

up_to loop 0 1

main up_to b
 unsafe b resume match
 yield

Future

main関数のgen関数のresumeで、各タスクの再開を行います。
11-68図

11-68図 11-67図の続き

```
resume 0 : Pending
resume 1 : Pending
resume 2 : Ready
```

タスクの状態は、**Pending** から **Ready** になります。

Future は、タスクの実行結果を返すオブジェクトです。Futureは“
”タスクの実行結果を返すオブジェクトです。Futureは“
”タスクの実行結果を返すオブジェクトです。**Future** は **Pending** から
タスクの実行結果を返すオブジェクトです。**Ready** になると、
タスクの実行結果を返すオブジェクトです。

Semi-Coroutineは、Futureと似た概念です。
Futureは、タスクの実行結果を返すオブジェクトです。

11.3.2 Future

Futureは、タスクの実行結果を返すオブジェクトです。
HTTPリクエストの実行結果を返すFutureオブジェクトです。

RustのFutureは、futures-rsライブラリで定義されています。

- **Future** タスクの実行結果を返すオブジェクト
- **Executor** タスクの実行を管理するオブジェクト
- **Task** タスクの実行単位

futures-rsライブラリは、Futureの実装を提供します。

Future

Rust 的 Future 是一个 trait，定义在 11-69

11-69 Future trait

```
1. pub trait Future {  
2.     type Output;  
3.     fn poll(self: Pin<&mut Self>, lw: &LocalWaker)  
4.         -> Poll<Self::Output>;  
5. }
```

11-69 的 std::future 模块 Future trait [6] 定义了一个 Output 的 poll

poll 是 Future 的一个方法，它接受一个 Pin 和 LocalWaker，返回一个 Rust 的 Future 的 poll 方法返回一个 Poll 类型，定义在 11-70

11-70 Poll<T>

```
1. pub enum Poll<T> {  
2.     Ready(T),  
3.     Pending,  
4. }
```

11-70 的 Poll<T> 枚举类型，Ready<T> 和 Pending 是 Option<T> 和 Result<T, E> 的变体，Future 的 poll 方法返回 Future 的 Future 类型

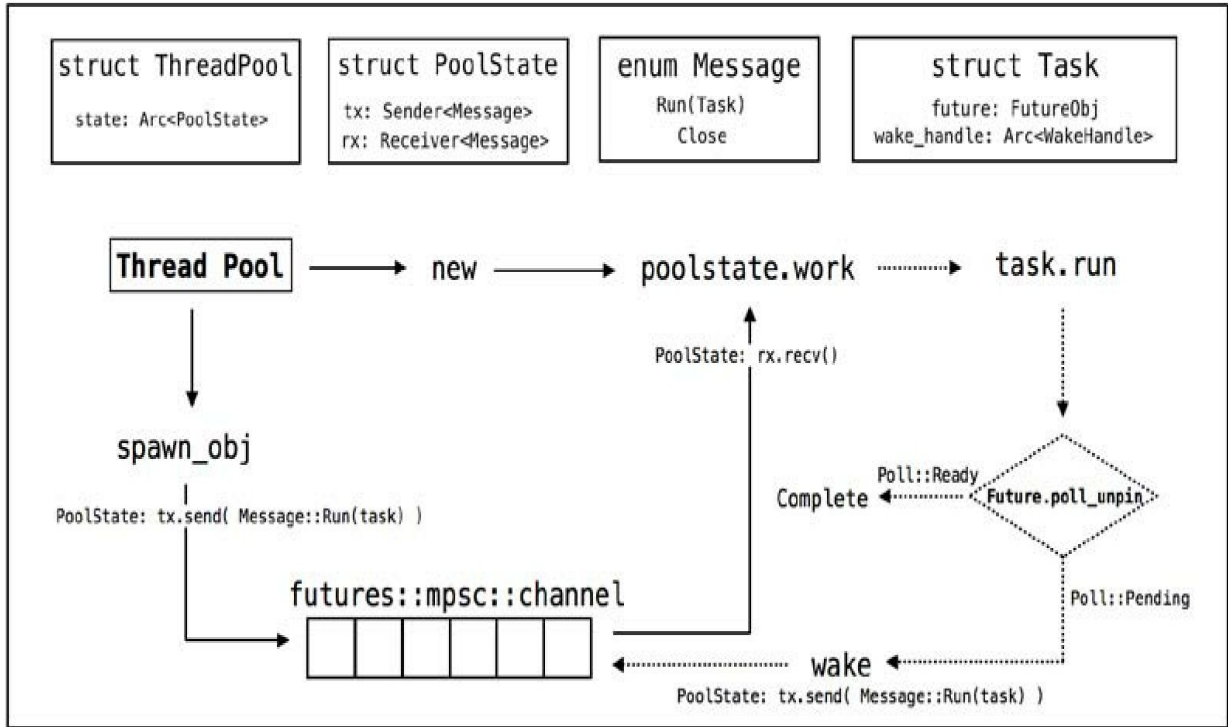
Executor 和 Task

Future 的 poll 方法返回 Poll<T>，Executor 和 Task 是

定义在 futures-rs 库中，Task 是 futures-rs [7] 定义，Executor 定义在 11-13

futures-rs 库中 crate 定义 futures-executor 定义 Executor

11-13 futures-rs ThreadPool PoolState Message Task



11-13 futures-rs Executor Task

ThreadPool state Arc PoolState

PoolState tx rx Sender Message Receiver Message std sync mpsc Channel futures-channel tx rx Channel

Message Run Task Message Channel Task

Task future wake_handle FutureObj Arc WeakHandle FutureObj Future futures-executor Future Future trait WeakHandle

```

11-13 Executor
Executor Channel ThreadPool spawn_obj Task send Channel spawn_obj PoolState tx Message Run task Channel
ThreadPool new PoolState work Channel work PoolState rx Message Run task
spawn_obj Channel work
work Message Run task Task run task run task FutureObj poll_unpin poll Pending Ready Pending task WakeHandle Channel Ready
futures-rs 11-13 Rust Future

```

11.3.3 async/await

```

futures-rs 0.1 then and_then Future async/await 0.3
11-71 futures-rs
11-71 futures-rs

```



```

1. // futures-rs 0.1
2. fn download_and_write_tweets(
3.     user: String,
4.     socket: Socket,
5. ) -> impl Future<Output = io::Result<()>> {
6.     pull_down_tweets(user)
7.         .and_then(move |tweets| write_tweets(socket))
8. }
9. // futures-rs 0.3
10. async fn download_and_write_tweets(
11.     user: &str,
12.     socket: &Socket,
13. ) -> io::Result<()> {
14.     let tweets = await!(pull_down_tweets(user))?;
15.     await!(write_tweets(socket))
16. }

```

11-71 `download_and_write_tweets` `pull_down_tweets`
`write_tweets` `and_then`
`async` `await` `and_then`

Rust `async` `await` `async/await`
`await`

`async/await` `async fn` `impl`
`Future` 11-71

async/await

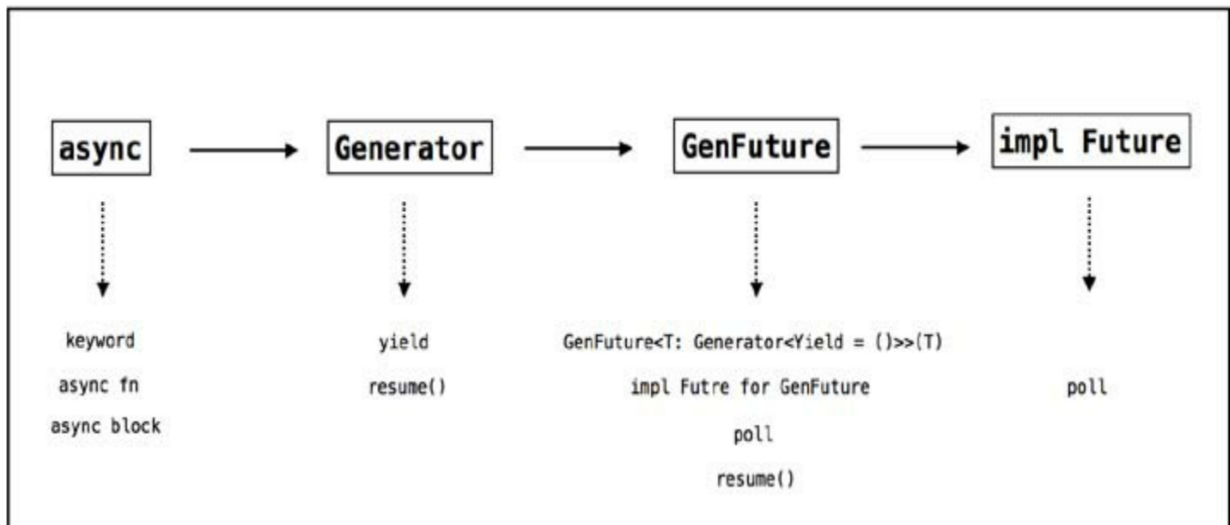
Rust `async fn` `async` 11-72
 11-72 **async**

```

1. let my_future = async {
2.     await!(prev_async_func);
3.     println!("Hello from an async block");
4. };

```

11-72 async Future async fn
 async Future 11-14



11-14 async Future

11-14 async Rust
 AST HIR async async
 Generator Yield = GenFuture
 GenFuture T Generator Yield = T
 GenFuture Future 11-73

11-73 GenFuture Future

```

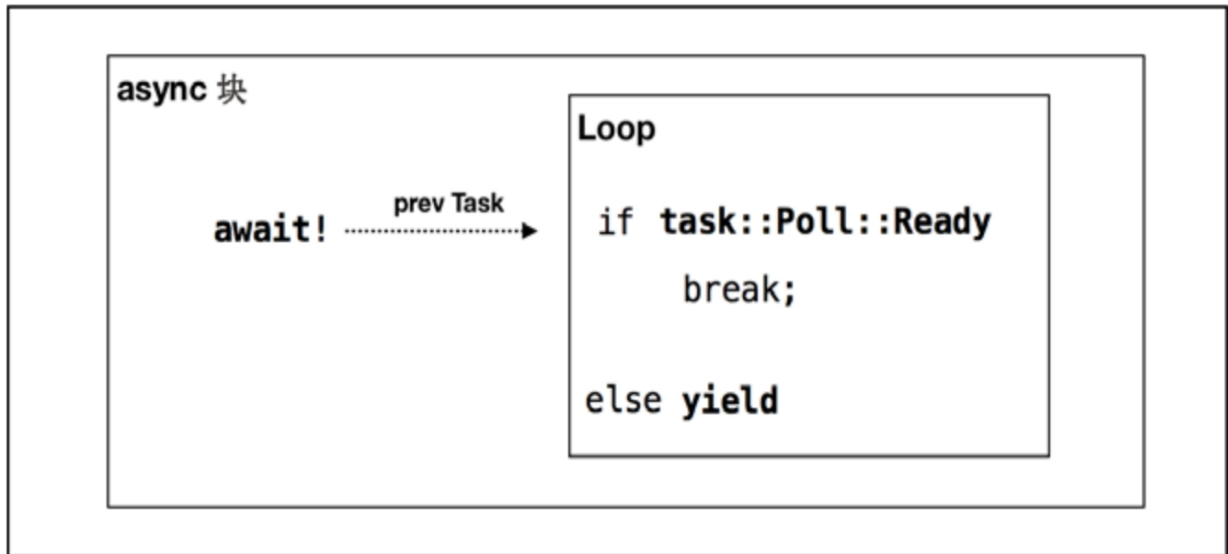
1. impl<T: Generator<Yield = ()>> Future for GenFuture<T> {
2.     type Output = T::Return;
3.     fn poll(self: Pin<&mut Self>, lw: &LocalWaker)
4.         -> Poll<Self::Output>
5.     {
6.         set_task_waker(lw, ||
7.             match unsafe { Pin::get_mut_unchecked(self).0.resume() }
8.             {
9.                 GeneratorState::Yielded(()) => Poll::Pending,
10.                GeneratorState::Complete(x) => Poll::Ready(x),
11.            }
12.        )
13.    }
14. }

```

11-73 std::future::GenFuture Future
 poll resume Pin get_mut_unchecked
 self &mut self "&mut self.0.resume"
 resume Future 11-67 Future

std::future::from_generator Future
 GenFuture

async GenFuture
 await await 11-15



11-15 await

await async await loop Ready loop Pending yield yield async Generator Yield=

Pin UnPin

Pin Pin T std pin Rust 2018 Rust 1.30 Pin T

11-62 gen 11-74

11-74 11-62

```

1. let mut gen = ||{
2.     let x = 1u64;
3.     let ref_x = &x;
4.     yield 1;
5.     yield 2;
6.     yield 3;
7.     return 4;
8. };
  
```

11-74 `x` `ref_x` `ref_x` `x`
error[E0626]: borrow may still be in use when generator yields

```
|     let ref_x = &x;  
|           ^  
|     ...  
|     ----- possible yield occurs here
```

Rust
11-65
Self-referential Struct 11-75
11-74
11-75 **11-74**

```

1.  enum __Gen<'a> {
2.      Start,
3.      State1(State1<'a>),
4.      State2(State2),
5.      State3(State3),
6.      Done
7.  }
8.  struct State1<'a> { x: u64, ref_x: &'a u64 }
9.  impl<'a> Generator for __Gen<'a> {
10.     type Yield = u64;
11.     type Return = u64;
12.     unsafe fn resume(&mut self) -> GeneratorState<u64, u64> {
13.         match std::mem::replace(self, __Gen::Done) {
14.             __Gen::Start => {
15.                 let x = 1;
16.                 let statel = State1{x: x, ref_x: &x};
17.                 *self = __Gen::State1(statel);
18.                 GeneratorState::Yielded(1)
19.             }
20.             __Gen::State1(State1{x: 1, ref_x: &1}) => {
21.                 *self = __Gen::State2(State2{x: 2});
22.                 GeneratorState::Yielded(2)
23.             }
24.             // ...省略
25.         }
26.     }
27. }

```

11-75 State1
 8 ref_x x

resume 15 17 resume
 State1 State2 20 match
 replace State1

replace State1 State2
 State1 State1 x

```
ref_xxxxxxxxxxxxxx Rustxxxxxxxxxxxxxxxxxx  
resumeunsafe
```

Box T Arc T Rust Rust Pin T

PinT Dereference
PinT 11-76

11-76 Pin T

```
1.  #![feature(pin)]
2.  use std::pin::{Pin, Unpin};
3.  use std::marker::Pinned;
4.  use std::ptr::NonNull;
5.  struct Unmovable {
6.      data: String,
7.      slice: NonNull<String>,
8.      _pin: Pinned,
9.  }
10. impl Unpin for Unmovable {}
11. impl Unmovable {
12.     fn new(data: String) -> Pin<Box<Self>> {
13.         let res = Unmovable {
14.             data,
15.             slice: NonNull::dangling(),
16.             _pin: Pinned,
17.         };
18.         let mut boxed = Box::pinned(res);
19.         let slice = NonNull::from(&boxed.data);
20.         unsafe {
21.             let mut_ref: Pin<&mut Self> = Pin::as_mut(&mut boxed);
22.             Pin::get_mut_unchecked(mut_ref).slice = slice;
23.         }
24.         boxed
25.     }
26. }
27. fn main() {
28.     let unmoved = Unmovable::new("hello".to_string());
29.     let mut still_unmoved = unmoved;
30.     assert_eq!(still_unmoved.slice,
31.         NonNull::from(&still_unmoved.data));
32.     let mut new_unmoved = Unmovable::new("world".to_string());
33.     std::mem::swap(&mut *still_unmoved, &mut *new_unmoved);
34. }
```


11-76 1 [feature pin] Pin
T

2 Pin Unpin Pin “” Rust Pin T
 “” Unpin Pin “”
 Unpin

3 Pinned std marker
 Pinned Unpin

4 NonNull T

5 9 Unmovable slice
 data Unmovable Pinned Unpin

10 Unmovable Unpin

11 26 Unmovable new Pin Box Self
 new Unmovable res Box
 pinned res Pin Box Unmovable boxed
 NonNull from boxed data NonNull slice
 unsafe Pin as_mut &mut boxed Pin
&mut Self mut_ref Pin &mut
 Unmovable Pin get_mut_unchecked Pin &mut
 Unmovable &mut Unmovable slice slice

main new Unmovable unmoved
 still_unmoved unmoved 30 31
 slice data
 data Pin T

32 33 Unmovable new_unmoved
 std mem swap still_unmoved
 10 Unmovable Unpin 10 swap
 Unmovable Pinned Unpin
 swap

11-73 GenFuture poll Pin &mut
 Self

`Pin::get_mut_unchecked` 和 `Pin::T` 的 API 文档

async/await

Rust 社区对 `futures-rs` 库进行了封装，提供了 `Future` 和 `Task` 两个 trait，以及 `async/await` 宏。通过 `futures-rs` 库，我们可以使用 `Future` 和 `Task` 来编写异步代码，而无需使用 `API`。

在 Rust 中，我们可以使用 `cargo new` 命令来创建一个新的 crate，名为 `“futures-demo”`。在 `Cargo.toml` 文件中，我们可以指定使用 `futures-preview` 版本 `0.3.0-alpha.7`。在 `main.rs` 文件中，我们可以编写以下代码：

11-77 async/await

```
1.  #![feature(arbitrary_self_types, futures_api)]
2.  #![feature(async_await, await_macro, pin)]
3.  use futures::{
4.      executor::ThreadPool,
5.      task::SpawnExt,
6.  };
7.  use std::future::Future;
8.  use std::pin::Pin;
9.  use std::task::*;
10. pub struct AlmostReady {
11.     ready: bool,
12.     value: i32,
13. }
14. pub fn almost_ready(value: i32) -> AlmostReady {
```

```

15.     AlmostReady { ready: false, value }
16. }
17. impl Future for AlmostReady {
18.     type Output = i32;
19.     fn poll(self: Pin<&mut Self>, lw: &LocalWaker) ->
20.         Poll<Self::Output>
21.     {
22.         if self.ready {
23.             Poll::Ready(self.value + 1)
24.         } else {
25.             unsafe { Pin::get_mut_unchecked(self).ready = true ;}
26.             lw.wake();
27.             Poll::Pending
28.         }
29.     }
30. }
31. fn main() {
32.     let mut executor = ThreadPool::new().unwrap();
33.     let future = async {
34.         println!("howdy!");
35.         let x = await!(almost_ready(5));
36.         println!("done: {:?}", x);
37.     };
38.     executor.run(future);
39. }

```

11-77 [feature
arbitrary_self_types futures_api] [feature
async_awaitawait_macropin]

36futures-rsexecutorThreadPooltaskSpawnExtFutureexecutortask

79Future Pin task
async/await

```
    10 13  AlmostReady bool ready i32
    value
```

```
    14 16  almost_ready i32
    ready false AlmostReady
```

```
    17 30  AlmostReady Future poll
    Pin &mut Self lw &LocalWaker
    Executor poll
    AlmostReady ready true
    true Poll Ready self.value+1 false
    ready true lw wake
    11-13 Executor Task Poll Pending
```

```
    main
    32 ThreadPool new executor
    33 37  async Future future async
    almost_ready AlmostReady await
```

```
    38  executor run future 11-
    13  Executor Task run future
    FutureObj spawn_obj Channel work
```

```
    futures-demo
    11-78
```

11-78 async/await

howdy!

done: 6

· Future

· Task

Task 对象可以看作“线程”的抽象，futures.rs 模块的 spawn_obj 函数返回 Future 对象，Future 对象可以看作 Future 类型的值。

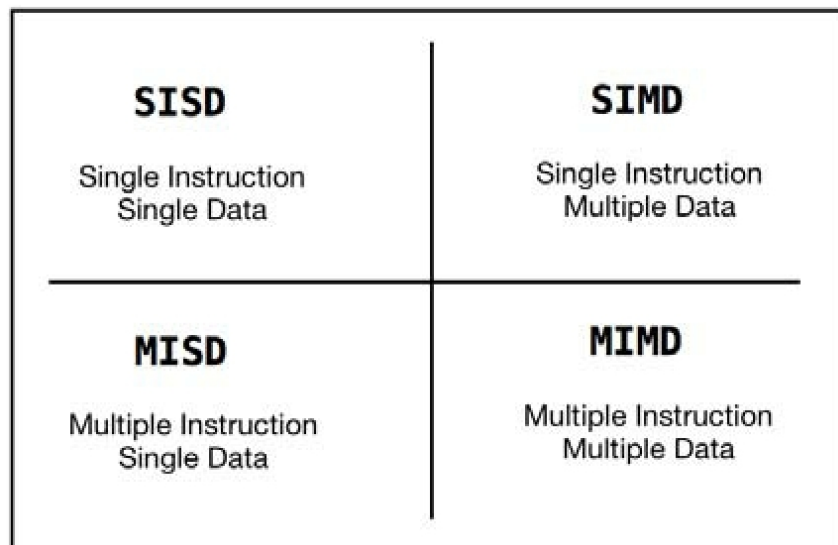
futures.rs 模块的 Future 对象可以看作 Future 类型的值，可以调用 Future 对象的 wait 方法。

11.4 并行

并行是指同时执行多个任务，在多线程编程中，并行是指同时执行多个线程。并行可以分为任务并行和数据并行。

任务并行 (Task Parallelism) 是指将任务分解为多个子任务，每个子任务由一个线程执行。数据并行 (Data Parallelism) 是指将数据分解为多个子数据，每个子数据由一个线程执行。

Flynn 模型将并行分为 11-16 种。

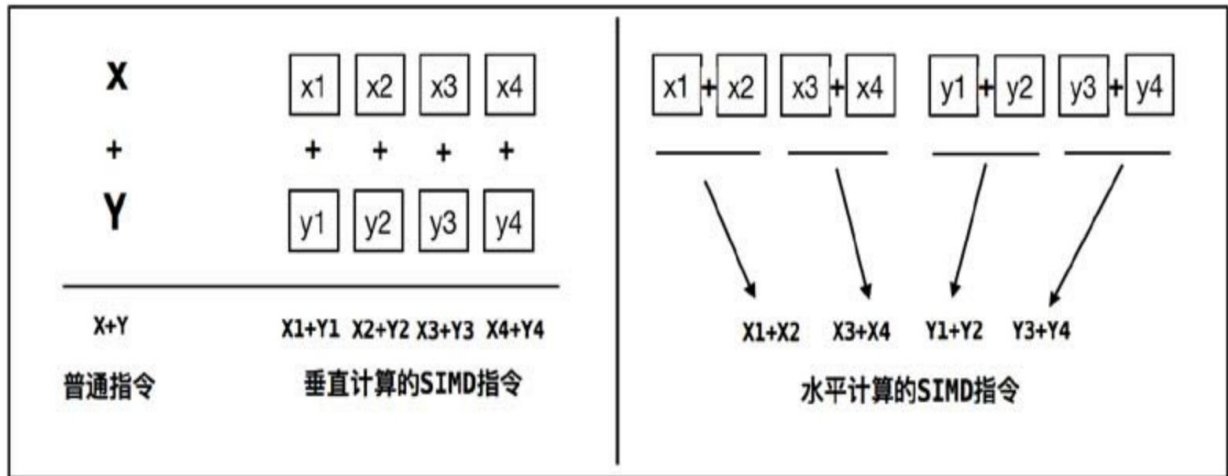


11-16 种并行模型

SISD 是指单指令单数据，即只有一个 CPU 执行一条指令，处理单一数据。

MISD 是指多指令单数据，即多个 CPU 执行多条指令，处理单一数据。

普通指令和SIMD指令的计算方式对比



11-17 SIMD指令集

AVX-256指令集支持256位宽的SIMD寄存器

Vectorization 是指将标量运算转换为SIMD指令的过程。SIMD指令可以同时处理多个数据元素，从而提高计算效率。SIMD指令集包括SSE、AVX-256、AVX-512等。

11.4.2 Rust SIMD

Rust 1.27版本开始支持SIMD指令集。x86_64架构支持SSE、SSE2、SSE3、SSSE3、AVX-256、AVX-512等指令集。

Rust通过std::arch模块提供SIMD支持。Rust SIMD支持x86_64、ARMv8-A等架构。std::arch模块提供了x86_64 SIMD指令集的不安全接口。

SIMD入门

创建新项目并添加crate: "simd-demo" Cargo.toml添加stdsimd依赖

0.1.0src/main.rs11-79

11-79SIMD

```
1.  #![feature(stdsimd)]
2.  use ::std as real_std;
3.  use stdsimd as std;
4.  #[cfg(target_arch = "x86")]
5.  use ::std::arch::x86::*;
6.  #[cfg(target_arch = "x86_64")]
7.  use ::std::arch::x86_64::*;
8.  fn main() {
9.      if is_x86_feature_detected!("sse4.2") {
10.         #[target_feature(enable = "sse4.2")]
11.         unsafe fn worker() {
12.             let needle = b"\r\n\t ignore this ";
13.             let haystack = b"Split a \r\n\t line ";
14.             let a = _mm_loadu_si128(needle.as_ptr() as *const _);
15.             let b = _mm_loadu_si128(haystack.as_ptr() as *const _);
16.             let idx = _mm_cmpestri(
17.                 a, 3, b, 20, _SIDD_CMP_EQUAL_ORDERED
18.             );
19.             assert_eq!(idx, 8);
20.         }
21.         unsafe { worker(); }
22.     }
23. }
```

11-79 [featurestdsimd] stdsimdNightly

2usestd"real_std"stdstdsimd3

4[cfgtarget_arch=x86] CPUx86stdarchx86

6CPUx86_64

main 9 if is_x86_feature_detected
sse4.2 CPU CPU
CPU SSE4.2

11 20 unsafe worker SIMD
SIMD unsafe

12 13 needle haystack
haystack needle

14 _mm_loadu_si128 _m128i
128 Intel
_mm_loadu_si128 needle

15 haystack

16 _mm_cmpestri a needle
3 haystack b
20 _SIDD_CMP_EQUAL_ORDERED
_mm_cmpestri
haystack needle

19 8

cargo run CPU
SSE4.2

Rust LLVM src
auto_vector.rs 11-80

11-80 LLVM AVX2

```

1. fn add_quickly_fallback(a: &[u8], b: &[u8], c: &mut [u8]) {
2.     for ((a, b), c) in a.iter().zip(b).zip(c) {
3.         *c = *a + *b;
4.     }
5. }
6. #[cfg(any(target_arch = "x86", target_arch = "x86_64"))]
7. #[target_feature(enable = "avx2")]
8. unsafe fn add_quickly_avx2(a: &[u8], b: &[u8], c: &mut [u8]) {
9.     add_quickly_fallback(a, b, c)
10. }
11. fn add_quickly(a: &[u8], b: &[u8], c: &mut [u8]) {
12.     #[cfg(any(target_arch = "x86", target_arch = "x86_64"))]
13.     {
14.         if is_x86_feature_detected!("avx2") {
15.             println!("support avx2");
16.             return unsafe { add_quickly_avx2(a, b, c) }
17.         }
18.     }
19.     add_quickly_fallback(a, b, c)
20. }
21. fn main() {
22.     let mut dst = [0, 2];
23.     add_quickly(&[1, 2], &[2, 3], &mut dst);
24.     assert_eq!(dst, [3, 5]);
25. }

```

11-8015add_quickly_fallback

6 10 add_quickly_avx2 [cfg any
 target_arch= x86 target_arch= x86_64]
 [target_featureenable=avx2] LLVM
 x86x86_64AVX2
 add_quickly_fallback

```

    11-20 crate::add_quickly {
        #[cfg(any(
            target_arch = "x86",
            target_arch = "x86_64"
        ))]
        x86_x86_64 {
            if is_x86_feature_detected!("avx2") {
                add_quickly_avx2 {
                    AVX2 {
                        15 {
                            // ...
                        }
                    }
                }
            }
        }
    }

```

```

    x86_x86_64 {
        add_quickly_fallback {
            // ...
        }
    }
    main {
        add_quickly {
            // ...
        }
    }

```

Cargo.toml
 src
 main.rs
 auto_vector.rs
 main
 Cargo.toml
 bin
 11-81

11-81 Cargo.toml bin

```

[[bin]]
path = "src/auto_vector.rs"
name = "auto_vector"

[[bin]]
path = "src/main.rs"
name = "main"

```

crate main

cargo run --bin auto_vector
 auto_vector.rs
 main.rs
 cargo run --bin main

SIMD

11-79 SIMD
 x86

- **__m128i** 128-bit integer
- **__m128** 128-bit float (4x f32)
- **__m128d** 128-bit double (2x f64)
- **__m256i** 256-bit integer
- **__m256** 256-bit float (8x f32)
- **__m256d** 256-bit double (4x f64)

ARM

· **float32x2_t** 642f32

· **float32x4_t** 1282f32

· **int32x2_t** 642i32

· **int32x4_t** 1282i32

stdarchx86_mm256_add_epi64
_mm256_AVXaddmulabs
_pd64_ps32_epi32
32

stdsimdRustsimd
fasterimdeezstdsimd

faster
11-80

11-82faster

```
1. use faster::*;
2. fn main() {
3.     let two_hundred = (&[2.0f32; 100][..]).simd_iter()
4.         .simd_reduce(f32s(0.0), f32s(0.0), |acc, v| acc + v)
5.         .sum();
6.     assert_eq!(two_hundred, 200.0f32);
7. }
```

11-82fasterSIMD

11.5

CPU
Rust

```

Rust

```

Rust is a systems programming language that runs fast, is safe, easy to use, and fun.
 C++11 is a systems programming language that runs fast, is safe, easy to use, and fun.

Rust

```

    Rust
    async/await Rust
    async/await Rust
    Rust
    Rust

```

rust-simd
Nightly

□□□□□□□□ Rust □□□□□□□□□□□□□□□□□□□

- [1] [Linuxのtaskを動かす](#)
- [2] [play.rust-lang.org](#)
- [3] tx rx Transimt rx Receive x
- [4] [threadpool](https://crates.io/crates/threadpool) <https://crates.io/crates/threadpool>
- [5] <https://github.com/rayon-rs/rayon>
- [6] Rust Nightly 1.30
- [7] futures-rs 0.3

12

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

Meta-Programming Meta “”
Meta-Knowledge “” Meta-Cognition “” Meta-Programming
“”
Meta “”
“”

[illegible]

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □

- `package` `com.coder` `C/C++` `package` `com.coder`
- `package` `com.coder` `C/C++` `package` `com.coder`
- `package` `com.coder` `Ruby` `Java` `Go` `Rust` `package` `com.coder`

--	--	--	--	--	--

- `enum` `Ruby` `Elixir` `Rust` `...`

11

- □□□□□□ □□Go□□□go generate□□□□□□□□□□□□□□

```

    AST
    Rust derive
    trait

```

DSL
Rust AST

12.1 类型

通过 `Reflect` 模块可以方便地操作 Rust 的 `std::any::Any` 类型。

12-1 定义 `Any` 类型

12-1 `Any` 类型

```
1. pub trait Any: 'static {
2.     fn get_type_id(&self) -> TypeId;
3. }
4. impl<T: 'static + ?Sized> Any for T {
5.     fn get_type_id(&self) -> TypeId { TypeId::of::<T>() }
6. }
```

12-1 中 1-3 行定义了 `Any` 类型，它是一个 `static` 的 trait。4-6 行是 Rust 的 `static` 类型，它们实现了 `Any` 类型。

`get_type_id` 返回 `TypeId`，它是 Rust 中的一个类型标识符。它返回一个 `TypeId`，它是一个字符串，表示类型的唯一标识符。它返回一个 `TypeId`，它是一个字符串，表示类型的唯一标识符。

`Any` 类型可以用于检查一个类型是否是 `Any` 类型。12-2 定义

12-2 `Any` 类型的 `is` 方法

```
1. impl Any{
2.     pub fn is<T: Any>(&self) -> bool {
3.         let t = TypeId::of::<T>();
4.         let boxed = self.get_type_id();
5.         t == boxed
6.     }
7. }
```

12-2 中 `Any` 类型的 `is` 方法，它返回一个 `bool` 值，表示当前类型是否是 `Any` 类型。它返回一个 `bool` 值，表示当前类型是否是 `Any` 类型。

3 行 `TypeId::of::<T>()` 返回一个 `TypeId`，它是一个字符串，表示类型的唯一标识符。

12-4 self.get_type_id boxed
12-1 get_type_id TypeId of
5 t boxed bool

12.1.1 is

12-3 is

12-3 Any is

```
1. use std::any::{Any, TypeId};
2. enum E { H, He, Li}
3. struct S { x: u8, y: u8, z: u16 }
4. fn main() {
5.     let v1 = 0xc0ffee_u32;
6.     let v2 = E::He;
7.     let v3 = S { x: 0xde, y: 0xad, z: 0xbeef };
8.     let v4 = "rust";
9.     let mut a: &Any;
10.    a = &v1;
11.    assert!(a.is::<u32>());
12.    println!("{:?}", TypeId::of::<u32>());
13.    a = &v2;
14.    assert!(a.is::<E>());
15.    println!("{:?}", TypeId::of::<E>());
16.    a = &v3;
17.    assert!(a.is::<S>());
18.    println!("{:?}", TypeId::of::<S>());
19.    a = &v4;
20.    assert!(a.is::<&str>());
21.    println!("{:?}", TypeId::of::<&str>());
22. }
```

12-3 2 3 E S main
is

5 8 v1 u32 v2 v3 v4


```

4. struct S { x: u8, y: u8, z: u16 }
5. fn print_any(a: &Any) {
6.     if let Some(v) = a.downcast_ref::<u32>() {
7.         println!("u32 {:x}", v);
8.     } else if let Some(v) = a.downcast_ref::<E>() {
9.         println!("enum E {:?}", v);
10.    } else if let Some(v) = a.downcast_ref::<S>() {
11.        println!("struct S {:x} {:x} {:x}", v.x, v.y, v.z);
12.    } else {
13.        println!("else!");
14.    }
15. }
16. fn main() {
17.     print_any(& 0xc0ffee_u32);
18.     print_any(& E::He);
19.     print_any(& S{ x: 0xde, y: 0xad, z: 0xbeef });
20.     print_any(& "rust");
21.     print_any(& "hoge");
22. }

```

12-5 print_any &Any
 print_any if let downcast_ref

main print_any
 trait Any 12-6

12-6

```

u32 c0ffee
enum E He
struct S de ad beef
else!

```

&Any Box Any 12-7

12-7 Box Any

```

1. use std::any::Any;
2. fn print_if_string(value: Box<Any>) {
3.     if let Ok(string) = value.downcast::<String>() {
4.         println!("String (length {}): {}", string.len(), string);
5.     }else{
6.         println!("Not String")
7.     }
8. }
9. fn main() {
10.    let my_string = "Hello World".to_string();
11.    print_if_string(Box::new(my_string));
12.    print_if_string(Box::new(0i8));
13. }

```

12-7 print_if_string Box<Any>
 Box<Any> 12-5 print_any

3 if let Box<Any> downcast
 String downcast Result

12-8

12-8

```

String (length 11): Hello World
Not String

```

12.1.3

Any 12-9

12-9 Any

```

1. use std::any::Any;
2. struct UnStatic<'a> { x: &'a i32 }
3. fn main() {
4.     let a = 42;
5.     let v = UnStatic { x: &a };
6.     let mut any: &Any;
7.     //any = &v; // 编译错误
8. }

```

12-9 编译错误 UnStatic a 编译错误
static 编译错误

main 6 编译错误 &Any any 7 UnStatic
 &v any 编译错误

编译错误 UnStatic 编译错误 12-10
 编译错误

12-10 编译错误 UnStatic

```

1. use std::any::Any;
2. struct UnStatic<'a> { x: &'a i32 }
3. static ANSWER: i32 = 42;
4. fn main() {
5.     let v = UnStatic { x: &ANSWER };
6.     let mut a: &Any;
7.     a = &v;
8.     assert!(a.is::<UnStatic>());
9. }

```

12-10 3 编译错误 ANSWER 编译错误
 &ANSWER 编译错误 main &ANSWER UnStatic v
 编译错误 UnStatic Any

6 编译错误 &Any a 7 编译错误 &v a 8
 is 编译错误

12.2

```
Rust [REDACTED] Rust [REDACTED] Macro [REDACTED]  
[REDACTED]  
Macro Expansion
```

12.2.1 □□

[illegible]

C++
`#define min(X,Y) ((X)<Y)?(X):(Y)`
`X<Y?"min12":min12"`
`12"`
`12"`
`C++`

12-11

12-11 Lisp

```
1. (defmacro one! (var)
2.   (list 'setq var 1)
3. )
4. (+ (one! x) 2) // 调用 one!
5. (+ (setq x 1) 2) // 宏展开
6. )
```

```

12-11 Lisp 1 3 defmacro
one 4 one S one
"setq x 1" S "+setq x 1 2"

```

S“1 2 3”Lisp
 12-11“+setq x 1 2”S
 12-1

Rust
 Serde
 derive
Serialize
Deserialize

• println
 assert_eq
 thread_local

• derive
 Debug
 cfg

Rust

12-12
 12-12
 unless

```

1. macro_rules! unless {
2.     ($arg:expr, $branch:expr) => ( if !$arg { $branch };;)
3. }
4. fn cmp(a: i32, b: i32) {
5.     unless!( a > b, {
6.         println!("{}", a, b);
7.     });
8. }
9. fn main() {
10.     let (a, b) = (1, 2);
11.     cmp(a, b);
12. }
```

12-12
 macro_rules
 unless

4
 8
 cmp
 unless
 a
 b
 a
 b
 "1
 2"

12-13

12-13

```
1. #[derive(new)]
2. pub struct Foo;
3. fn main(){
4.     let x = Foo::new();
5.     assert_eq!(x, Foo);
6. }
```

12-13 new derive new Foo new 1 2

main new Foo x

12-14

12-14

```
1. macro_rules! stringify { ($($t:tt)*) => ({ /* compiler built-in */ }) }
2. macro_rules! println {
3.     () => (print!("\n"));
4.     ($fmt:expr) => (print!(concat!($fmt, "\n")));
5.     ($fmt:expr, $($arg:tt)*) =>
6.         (print!(concat!($fmt, "\n"), $($arg)*));
7. }
```

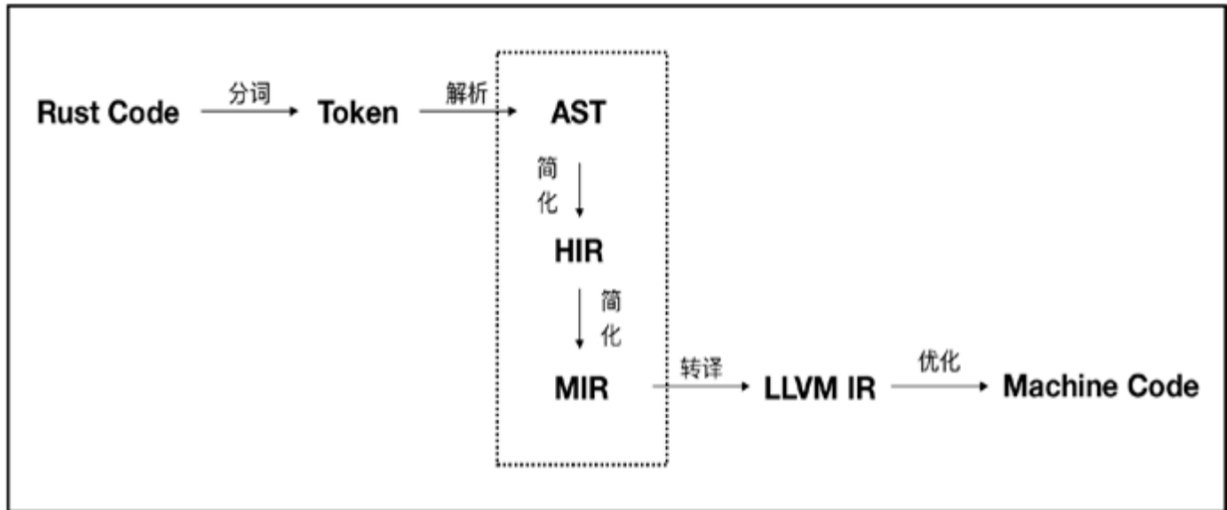
12-14 1 stringify

2 7 println concat print

Rust

12.2.3

Rust 12-2



12-2 Rust 编译过程

Rust 编译过程可以分为以下几个步骤 [\[1\]](#)

1. 将 Rust 源代码解析成 Token
2. 将 Token 解析成 AST
3. 将 AST 简化成 HIR (High-Level IR)
HIR 是 Rust 编译器内部的一种中间表示形式，它是在 AST 的基础上进行简化的，去掉了 AST 中的一些冗余信息，使得代码更加简洁。
4. 将 HIR 简化成 MIR (Middle-Level IR)
MIR 是 Rust 编译器内部的一种中间表示形式，它是在 HIR 的基础上进行简化的，去掉了 HIR 中的一些冗余信息，使得代码更加简洁。
5. 将 MIR 转译成 LLVM IR
6. 将 LLVM IR 优化成 Machine Code
LLVM IR 是 LLVM 编译器的一种中间表示形式，它是在 MIR 的基础上进行转译的，去掉了 MIR 中的一些冗余信息，使得代码更加简洁。

Rust 编译过程可以分为以下几个步骤 [\[1\]](#)
12-15 Rust 编译过程

12-15 Rust 编译过程

```

1. fn t(i: i32) -> i32{
2.     i + 2
3. }
4. fn main(){
5.     t(1);
6. }

```

12-15 Rust **main.rs** Cargo crate src/main.rs **rustc mian.rs cargo build**

- 函数
- 函数
- 函数
- 函数

t Identifier t i32

12-15 12-16

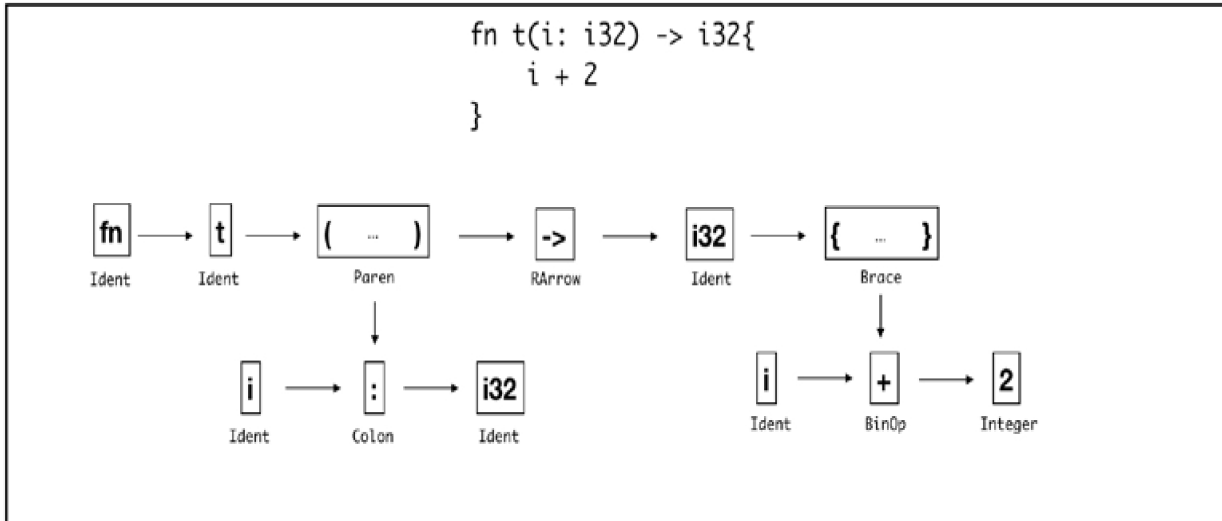
12-16 rustc

```

// 假如是单独的文件执行此命令
$ rustc -Z ast-json main.rs
// 假如是 cargo 生成的二进制包执行此命令
$ cargo rustc -- -Z ast-json

```

JSON AST 12-3 JSON



12-3

12-3 在 Rust 中，函数定义和函数调用是两种不同的语法结构。函数定义使用 `fn` 关键字，而函数调用则使用函数名。在 Rust 中，函数定义和函数调用是两种不同的语法结构。函数定义使用 `fn` 关键字，而函数调用则使用函数名。

在 Rust 中，函数定义和函数调用是两种不同的语法结构。函数定义使用 `fn` 关键字，而函数调用则使用函数名。

在 Rust 中，函数定义和函数调用是两种不同的语法结构。函数定义使用 `fn` 关键字，而函数调用则使用函数名。在 Rust 中，函数定义和函数调用是两种不同的语法结构。函数定义使用 `fn` 关键字，而函数调用则使用函数名。

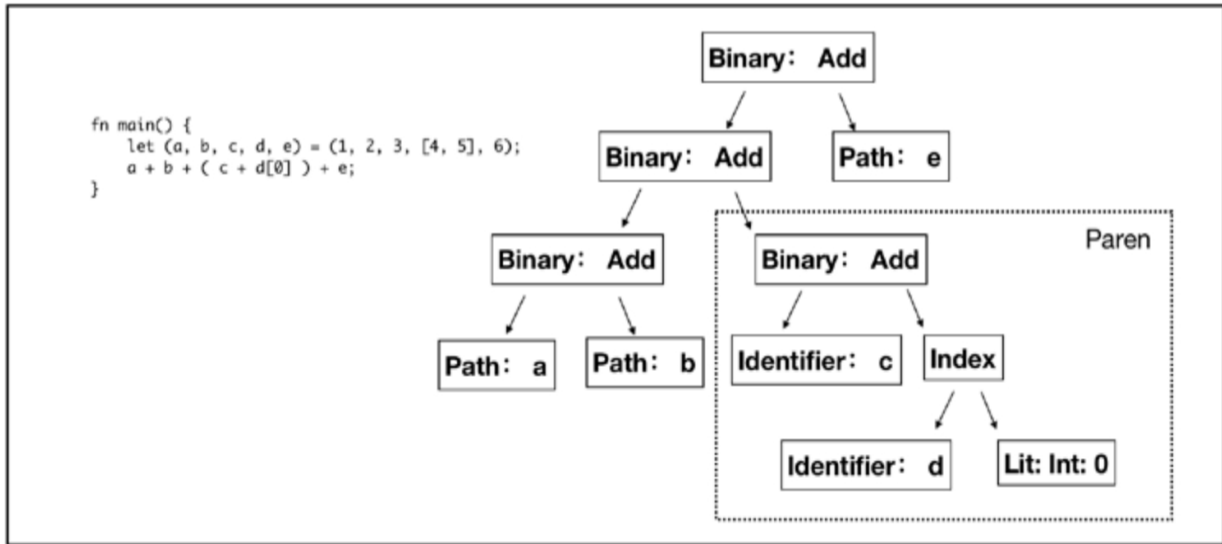
12-17 在 Rust 中，函数定义和函数调用是两种不同的语法结构。函数定义使用 `fn` 关键字，而函数调用则使用函数名。

12-17 在 Rust 中，函数定义和函数调用是两种不同的语法结构。函数定义使用 `fn` 关键字，而函数调用则使用函数名。

1. `fn main() {`
2. `let (a, b, c, d, e) = (1, 2, 3, [4, 5], 6);`
3. `a + b + (c + d[0]) + e;`
4. `}`

12-17 在 Rust 中，函数定义和函数调用是两种不同的语法结构。函数定义使用 `fn` 关键字，而函数调用则使用函数名。在 Rust 中，函数定义和函数调用是两种不同的语法结构。函数定义使用 `fn` 关键字，而函数调用则使用函数名。

12-4 在 Rust 中，函数定义和函数调用是两种不同的语法结构。函数定义使用 `fn` 关键字，而函数调用则使用函数名。



12-4

12-4 S 12-18

12-18 S

1. // a + b + (c + d[0]) + e
2. (
3. +
4. (
5. +
6. (+ a b)
7. (+ c (index d 0))
8.)
9. e
10.)

12-4 S 12-18

12.2.4

Rust macro_rules “Macro by example MBE”

macro_rules 12-19

12-19 macro_rules

```
1. macro_rules! $name {
2.     $rule0 ;
3.     $rule1 ;
4.     // ...
5.     $ruleN ;
6. }
```

12-19 \$name \$rule0 \$ruleN N 12-20

12-20

```
( $pattern ) => ( $expansion )
```

12-20 \$pattern \$expansion
12-12 unless
" \$arg expr \$branch expr " if
\$arg{\$branch} match

"\$arg expr" \$arg
"\$" expr

\$arg \$branch

12-21 unless

12-21 unless

```

1. macro_rules! unless {
2.     ($arg:expr, $branch:expr) => ( if !$arg { $branch });
3. }
4. fn main() {
5.     let (a, b) = (1, 2);
6.     unless!( a > b, {
7.         b - a;
8.     });
9. }

```

12-21 Rust main.rs

12-22

12-22

1. // 假如是单独的文件则执行此命令
2. \$ rustc -Z unstable-options --pretty=expanded main.rs
3. // 假如是 cargo 生成的二进制包则执行此命令
4. \$ cargo rustc -- -Z unstable-options --pretty=expanded

12-23

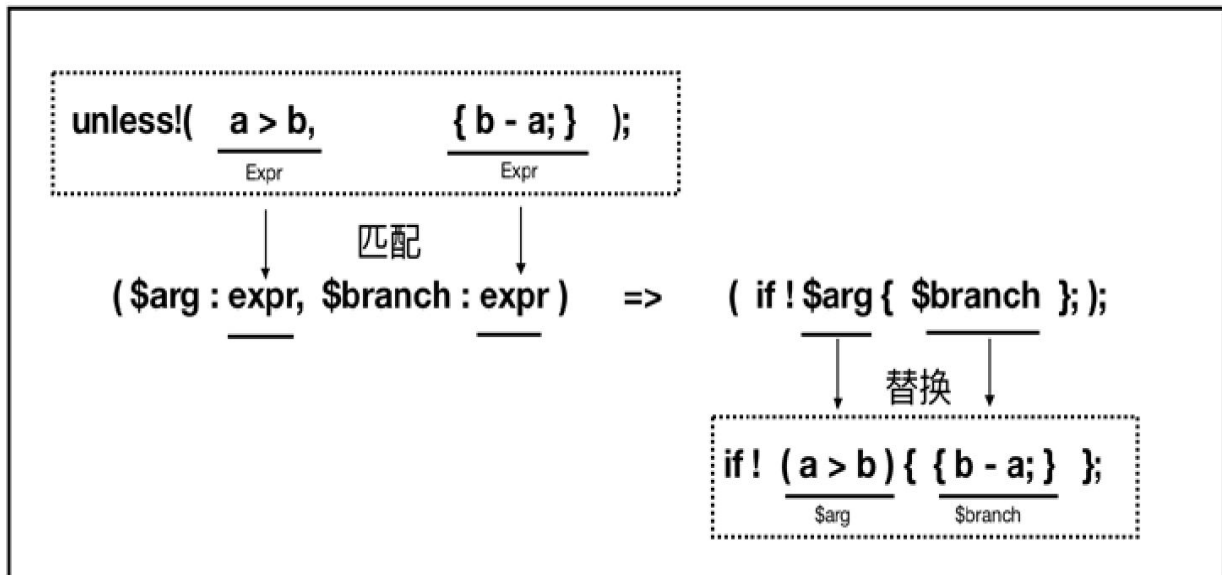
12-23

```

1. fn main() {
2.     let (a, b) = (1, 2);
3.     if !(a > b) { { b - a; } };
4. }

```

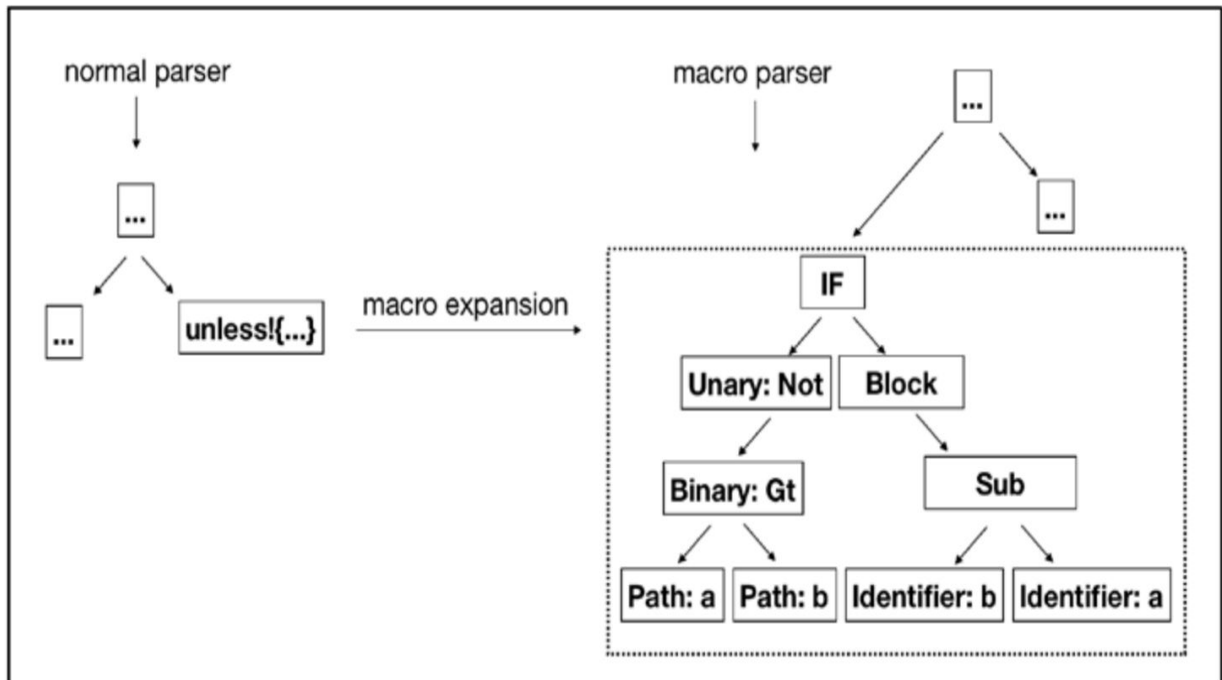
12-5



12-5 宏解析器

12-21 6 8 `unless` 12-5 宏解析器
`unless` 宏解析器 = 宏解析器
 宏解析器

宏解析器 `Normal Parser`
`Macro Parser`
 宏解析器
 12-6



12-6 AST

12-6 unless
 unless
 unless

12-24

12-24

```

1. fn macro_parser(
2.     sess: ParserSession,
3.     tts: TokenStream,
4.     ms: &[TokenTree]
5. ) -> NamedParseResult

```

12-24 macro_parser

- **sess** ParserSession
- **tts** TokenStream
- **ms**

macro_rules [2] 12-24

12-24

(\$lhs:tt) => (\$rhs:tt);+

macro_rules tt ms "+"

tts ms \$lhs 12-21 unless
"a b {b-a}" tts "\$arg
expr \$branch expr" "a b" "\$arg
expr" "{b-a}" "\$branch expr" \$arg \$branch
ms \$rhs \$rhs

ms
"+" "*"

[recursion_limit=
...]

expr

· item Rust impl

· block

· stmt

· expr

· pat

· ty

· ident

· path foo std iter

- **meta** [][][][][][][...][][][]
- **tt** []TokenTree[][][][][]
- **vis** [][][][]pub[]
- **lifetime** [][][][]

```

ttexpr

```

HashMap Rust 12-25

12-25 hashmap

```
1. fn main() {
2.     let map = hashmap!{
3.         "a" => 1,
4.         "b" => 2,
5.     };
6.     assert_eq!(map["a"], 1);
7. }
```

[illegible]

```

    "key="value"
"$key expr="value expr" key value Rust
Rust

```

“\$... sep rep”

- `$[...]` `XXXXXXXXXXXXXXXXXXXX`
- `sep [XXXXXXXXXXXXXXXXXXXXXXXXXXXX] = [XXXXXXXXXXXX]`

111

- [illegible]

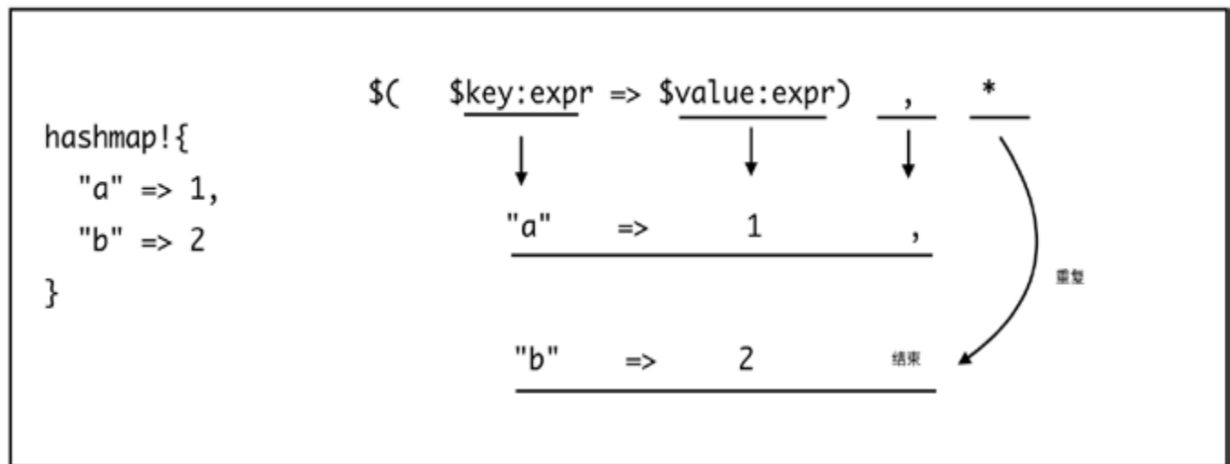
宏定义中，我们使用 `"$ $key:expr=$value:expr"` 来定义宏，其中 `$key:expr` 和 `$value:expr` 分别代表键和值，`*` 代表宏的结束符。

下面是一个使用 `hashmap` 宏的例子，来自第12-26节。

第12-26节 `hashmap` 宏的定义

```
1. macro_rules! hashmap {
2.     ($($key:expr => $value:expr),* ) => {
3.         {
4.             let mut _map = ::std::collections::HashMap::new();
5.             $(
6.                 _map.insert($key, $value);
7.             )*
8.             _map
9.         }
10.    };
11. }
```

下面是一个使用 `hashmap` 宏的例子，来自第12-26节。我们使用 `"$ $key:expr=$value:expr"` 来定义宏，其中 `$key:expr` 和 `$value:expr` 分别代表键和值，`*` 代表宏的结束符。



第12-7节 `hashmap` 宏的定义

第4节中，我们使用 `HashMap` 来存储键值对。在宏定义中，我们使用 `std::collections::HashMap` 来创建一个新的 `HashMap`。

第5节中，我们使用 `HashMap` 来存储键值对。在宏定义中，我们使用 `"$...$"` 来定义宏，其中 `$` 代表宏的结束符。

HashMap 12-27

12-27 HashMap

```
1. let mut _map = ::std::collections::HashMap::new();
2. _map.insert("a", 1);
3. _map.insert("b", 2);
4. _map
```

12-27

12-28

12-28 HashMap

```
1. macro_rules! hashmap {
2.     ($($key:expr => $value:expr,)* ) =>
3.         { hashmap!($($key => $value),*) };
4.     ($($key:expr => $value:expr),* ) => {
5.         {
6.             let mut _map = ::std::collections::HashMap::new();
7.             $(
8.                 _map.insert($key, $value);
9.             )*
10.            _map
11.        }
12.    };
13. }
```

12-28 2

12-29

12-29 使用宏定义 HashMap

```
1. macro_rules! hashmap {
2.     ($($key:expr => $value:expr),* $(,)* ) => {
3.         {
4.             let mut _map = ::std::collections::HashMap::new();
5.             $(
6.                 _map.insert($key, $value);
7.             )*
8.             _map
9.         }
10.    };
11. }
```

12-29 使用宏定义 HashMap 的语法如下：

```
macro_rules! hashmap {
    ($($key:expr => $value:expr),* $(,)* ) => {
        {
            let mut _map = ::std::collections::HashMap::new();
            $(
                _map.insert($key, $value);
            )*
            _map
        }
    };
}
```

使用 `hashmap` 宏定义 HashMap 的语法如下：

12-30 使用 `len` 方法获取字符串长度

12-30 使用 `len` 方法获取字符串长度

```
1. let _cap = <[()]>::len(&[()], ());
2. let mut _map = HashMap::with_capacity(_cap);
```

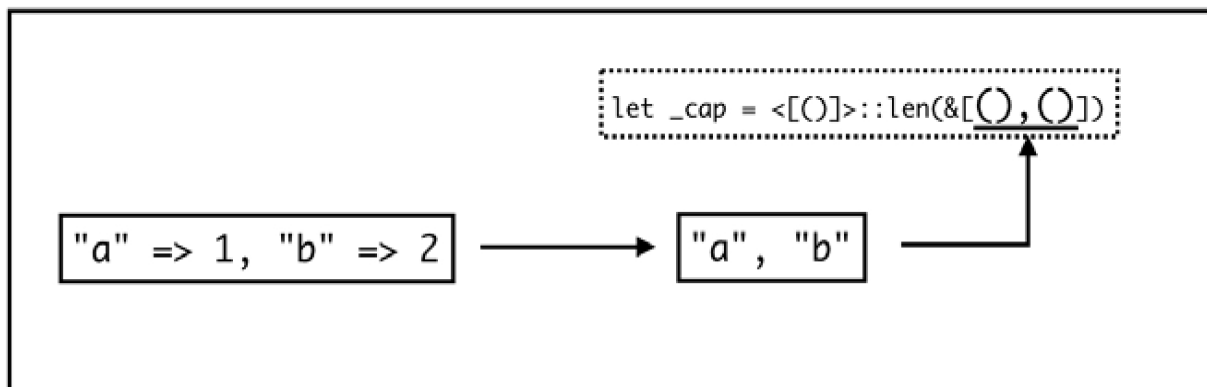
12-30 使用 `len` 方法获取字符串长度

`HashMap::with_capacity` 方法接受一个 `len` 参数，该参数是一个 `T` 类型的数组，该数组的长度即为 HashMap 的容量。

使用 `len` 方法获取字符串长度的语法如下：

使用 `len` 方法获取字符串长度的语法如下：

12-8



12-8 数组的容量

12-8 数组的容量

- 数组的容量
- 数组的容量
- 数组的容量

数组的容量 12-31 数组

12-31 数组 **hashmap**

```

1. macro_rules! unit {
2.     ($($x:tt)*) => (());
3. }
4. macro_rules! count {
5.     ($($key:expr),*) => (<[()]>::len(&[$(unit!($key)),*]));
6. }
7. macro_rules! hashmap {
8.     ($($key:expr => $value:expr),* $(,)* ) => {
9.         {
10.            let _cap = count!($($key),*);
11.            let mut _map
12.                = ::std::collections::HashMap::with_capacity(_cap);
13.            $(
14.                _map.insert($key, $value);
15.            )*
16.            _map
17.        }
18.    };
19. }
20. fn main(){
21.    let map = hashmap!{
22.        "a" => 1,
23.        "b" => 2,
24.    };
25.    assert_eq!(map["a"], 1);
26. }

```

12-31 unit count 12-8
 12-9

“@unit”“@count” [3] “@”
“unit”“unit”

· 12-32

· Nightly[feature_trace_macros]

12-33 12-32 hashmap

12-33 hashmap

```
1. #![feature(trace_macros)]
2. macro_rules! hashmap {
3.     (@unit $($x:tt)*) => (());
4.     (@count $($key:expr),*) =>
5.         (<[()]>::len(&[$(hashmap!(@unit $key)),*]));
6.     ($($key:expr => $value:expr),* $(,)* => ({
7.         let _cap = hashmap!(count! $($key),*);
8.         let mut _map =
9.             ::std::collections::HashMap::with_capacity(_cap);
10.         $(
11.             _map.insert($key, $value);
12.         )*
13.         _map
14.     });
15. }
16. fn main(){
17.     trace_macros!(true);
18.     let map = hashmap!{
19.         "a" => 1,
20.         "b" => 2,
21.     };
22. }
```

12-33 [feature=trace_macros] Nightly Rust 17 hashmap trace_macros true

12-34

12-34 hashmap

note: trace_macro

...

= note: expanding `hashmap! { "a" => 1 , "b" => 2 , }`

= note: to `{

let _cap = hashmap ! (count ! "a" , "b") ;

let mut _map = ::std::collections :: HashMap :: with_capacity (_cap) ;

_map . insert ("a" , 1) ;

_map . insert ("b" , 2) ; _map }`

= note: expanding `hashmap! { count ! "a" , "b" }`

= note: to `< [()] > :: len (

& [hashmap ! (unit ! "a") , hashmap ! (unit ! "b")])`

= note: expanding `hashmap! { unit ! "a" }`

= note: to `()`

= note: expanding `hashmap! { unit ! "b" }`

= note: to `()`

12-34 hashmap trace_macros

Hygienic Macro Rust 12-35

12-35

```

1. macro_rules! sum {
2.     ($e:expr) => ({
3.         let a = 2;
4.         $e + a
5.     })
6. }
7. fn main(){
8.     let four = sum!(a);
9. }

```

12-35

12-36

```

error[E0425]: cannot find value `a` in this scope
--> src/main.rs:8:22

```

```

|
8 |     let four = sum!(a);
|                                ^ not found in this scope

```

12-36

12-37

```

1. fn main(){
2.     let four = {
3.         let a = 2;
4.         a + a
5.     };
6. }

```

Rust

Rust

/

hashmap 使用 cargo 编译
hashmap_lite 使用 src/main.rs 编译
12-38

12-38 hashmap_lite

```
├─ Cargo.toml
├─ src
│   └─ lib.rs
│   └─ main.rs
```

hashmap 使用 src/lib.rs 编译
12-39

12-39 src/lib.rs

```
1. #[macro_export]
2. macro_rules! hashmap {
3.     // 同代码清单 12-32
4. }
```

12-39 使用 **[macro_export]**
hashmap 使用 src/main.rs 编译
12-40

12-40 src/main.rs

```
1. // Rust 2015
2. // #[macro_use] extern crate hashmap_lite;
3. // Rust 2018
4. use hashmap_lite::hashmap;
5. fn main(){
6.     let map = hashmap!{
7.         "a" => 1,
8.         "b" => 2,
9.     };
10.    assert_eq!(map["a"], 1);
11. }
```

12-40 Rust 2015 `[macro_use]` `extern crate` `hashmap_lite` `hashmap` `main` `hashmap` Rust 2015 `extern crate` `[macro_use]` Rust 2018 `use` `hashmap_lite` `hashmap` Rust 2018 `crate` `mod`

`[macro_use]` `mod` 12-41

12-41 `[macro_ues]` `mod`

```
1. #[macro_use]
2. mod macros {
3.     macro_rules! X { () => { Y!(); } }
4.     macro_rules! Y { () => {} }
5. }
6. fn main() {
7.     X!();
8. }
```

12-41 `X`

12-42

12-42 `mycrate` `incr`

```
1. pub fn incr(x: u32) -> u32 {
2.     x+1
3. }
4. #[macro_export]
5. macro_rules! inc {
6.     ($x:expr) => ( ::mycrate::incr($x) )
7. }
```

12-42 `mycrate` `incr` `inc` `mycrate` `incr $x` `mycrate` 12-43

12-43 `extern crate mycrate` `mc`

```

1.  #[macro_use]
2.  extern crate mycrate as mc;
3.  fn main(){ // ... }

```

12-43 Rust
\$crate 12-44

12-44 \$crate

```

1.  #[macro_export]
2.  macro_rules! inc {
3.      ($x:expr) => ( $crate::incr($x) )
4.  }

```

12-44 &crate 12-43 “mc incr \$x”
12-45

macro

Nightly Rust [feature decl_macro]
macro 12-45

12-45 macro

```

1.  #![feature(decl_macro)]
2.  macro unless($arg:expr, $branch:expr) {
3.      ( if !$arg { $branch });
4.  }
5.  fn cmp(a: i32, b: i32) {
6.      unless!( a > b, {
7.          println!("{}", a, b);
8.      });
9.  }
10. fn main() {
11.     let (a, b) = (1, 2);
12.     cmp(a, b);
13. }

```

图 12-45 展示了 `macro` 规则在 `unless` 宏规则中的应用。在 `macro_rules` 宏规则中，`macro` 规则在 Rust 2.0 版本中被引入，用于定义宏规则。在 `feature` 宏规则中，

12.2.5 小结

本章介绍了 Rust 宏系统的基本概念和用法。首先，我们介绍了宏的定义和调用。然后，我们介绍了宏的扩展和解析。最后，我们介绍了宏的注册和卸载。在 Rust 2.0 版本中，引入了 `feature` 宏规则，用于定义宏规则。在 `plugin_registrar` 宏规则中，

本章介绍了 Rust 宏系统的基本概念和用法。首先，我们介绍了宏的定义和调用。然后，我们介绍了宏的扩展和解析。最后，我们介绍了宏的注册和卸载。在 Rust 2.0 版本中，引入了 `feature` 宏规则，用于定义宏规则。在 `plugin_registrar` 宏规则中，

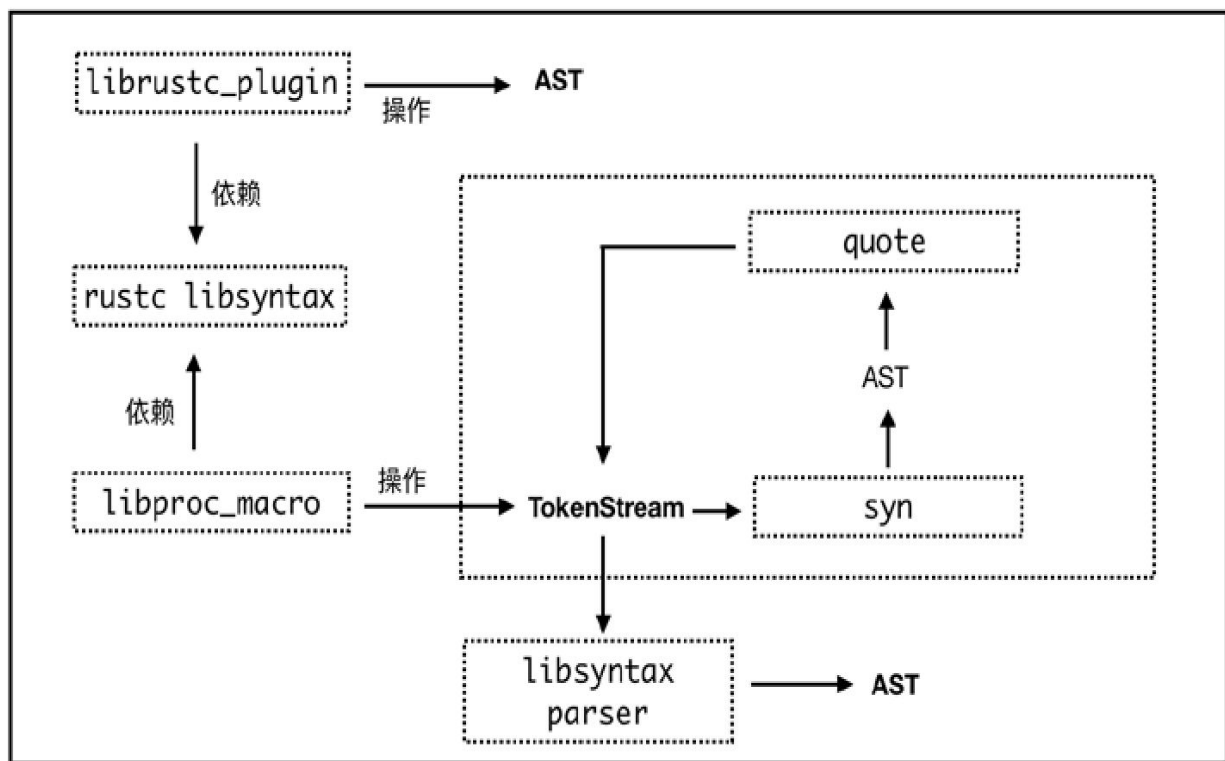
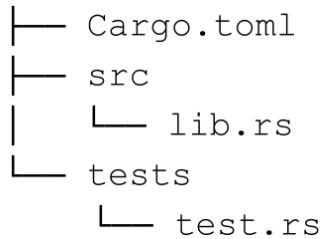


图 12-10 Rust 宏系统架构图

Rust 宏系统的基本概念和用法。首先，我们介绍了宏的定义和调用。然后，我们介绍了宏的扩展和解析。最后，我们介绍了宏的注册和卸载。在 Rust 2.0 版本中，引入了 `feature` 宏规则，用于定义宏规则。在 `plugin_registrar` 宏规则中，



Cargo.toml proc_macro 12-47

12-47 Cargo.toml lib proc_macro

```
[lib]
proc_macro = true
```

```
proc_macro lib

```

[TDD \[4\]](#)
[simple_proc_macro](#)
[tests/test.rs](#)
[12-48](#)

12-48 tests/test.rs

```
1. #[macro_use]
2. extern crate simple_proc_macro;
3. #[derive(A)]
4. struct A;
5. #[test]
6. fn test_derive_a() {
7.     assert_eq!("hello from impl A".to_string(), A.a());
8. }
```

src/lib.rs 12-49

12-49 src/lib.rs [deriue A]


```

1.  #![feature(custom_attribute)]
2.  #[proc_macro_attribute]
3.  pub fn attr_with_args(args: TokenStream, input: TokenStream)
4.      -> TokenStream {
5.      let args = args.to_string();
6.      let input = input.to_string();
7.      format!("fn foo() -> &'static str {{ {} {} }}", args)
8.          .parse().unwrap()
9.  }

```

```

12-51 1 [feature custom_attribute]
2 [proc_macro_attribute]
3 attr_with_args args
input TokenStream TokenStream args
[attr_with_args Hello Rust]
input foo
5 6 args input
7 format args foo
parse Result TokenStream Err unwrap
cargo test
Bang
macro_rules hashmap
hashmap 12-52
12-52 tests/test.rs hashmap

```

```

1.  #![feature(proc_macro_non_items)]
2.  use simple_proc_macro::hashmap;
3.  #[test]
4.  fn test_hashmap() {
5.      let hm = hashmap!{ "a": 1, "b": 2, };
6.      assert_eq!(hm["a"], 1);
7.      let hm = hashmap!{ "a" => 1, "b" => 2, "c" => 3 };
8.      assert_eq!(hm["d"], 4);
9.  }

```

```

12-52 [feature proc_macro_non_items]
Bang [macro_use] use
hashmap 5 7
src/lib.rs 12-53
12-53 src/lib.rs

```

```

1.  #![feature(proc_macro_non_items)]
2.  #[proc_macro]
3.  pub fn hashmap(input: TokenStream) -> TokenStream {
4.      let input = input.to_string();
5.      let input = input.trim_right_matches(',');
6.      let input: Vec<String> = input.split(",").map(|n| {
7.          let mut data = if n.contains(":") { n.split(":") }
8.                          else { n.split(" => ") };
9.          let (key, value) =
10.             (data.next().unwrap(), data.next().unwrap());
11.          format!("hm.insert({}, {})", key, value)
12.     }).collect();
13.     let count: usize = input.len();
14.     let tokens = format!("
15.         {{
16.         let mut hm =
17.             ::std::collections::HashMap::with_capacity({});
18.             {}
19.             hm
20.         }}", count,
21.         input.iter().map(|n| format!("{}", n)).collect::<String>()
22.     );
23.     tokens.parse().unwrap()
24. }

```

```

12-53 [feature proc_macro_non_items]
Nightly Rust [proc_macro]
hashmap Bang TokenStream

```

HashMap 的输入输出格式
format 解析

Bang macro_rules

proc_macro TokenNode TokenTree
TokenStream quote

syn quote

proc_macro Rust syn
quote proc_macro2
serde syn quote serde
serde

syn Rust quote syn
proc_macro TokenStream proc_macro syn quote
derive-new [5]

cargo new derive-new tests/test.rs

12-54 derive-new

```
├── Cargo.toml
├── src
│   └── lib.rs
└── tests
    └── test.rs
```

Cargo.toml 12-55

12-55 Cargo.toml

```
[lib]
proc-macro = true
[dependencies]
```

```
quote = "0.6"
syn = "0.15"
proc-macro2="0.4"
```

12-55 在 `proc-macro` 库中定义 `quote` 宏，版本 0.6，`syn` 版本 0.15，`syn` 宏定义 `quote` 宏，版本 0.12，API 文档

TDD 测试 `derive-new` 宏，`new` 宏在 `tests/test.rs` 中定义，12-56

12-56 `tests/test.rs` 中的代码

```
1. use derive_new::New;
2. // 无字段结构体
3. #[derive(New, PartialEq, Debug)]
4. pub struct Foo {}
5. // 包含字段的结构体
6. #[derive(New, PartialEq, Debug)]
7. pub struct Bar {
8.     pub x: i32,
9.     pub y: String,
10. }
11. // 单元结构体
12. #[derive(New, PartialEq, Debug)]
13. pub struct Baz;
14. // 元组结构体
15. #[derive(New, PartialEq, Debug)]
16. pub struct Tuple(pub i32, pub i32);
```

12-56 在 `derive` 宏中定义 `New`、`PartialEq`、`Debug` 宏，`New` 宏定义 `new` 宏，`PartialEq` 宏定义 `PartialEq` 宏，`Debug` 宏定义 `Debug` 宏，`Rust` 宏定义 `Rust` 宏

在 `new` 宏中定义 `new` 宏，12-57

12-57 `tests/test.rs` 中的 `new` 宏

```

1.  #[test]
2.  fn test_empty_struct() {
3.      let x = Foo::new();
4.      assert_eq!(x, Foo {});
5.  }
6.  #[test]
7.  fn test_simple_struct() {
8.      let x = Bar::new(42, "Hello".to_owned());
9.      assert_eq!(x, Bar { x: 42, y: "Hello".to_owned() });
10. }
11. #[test]
12. fn test_unit_struct() {
13.     let x = Baz::new();
14.     assert_eq!(x, Baz);
15. }
16. #[test]
17. fn test_simple_tuple_struct() {
18.     let x = Tuple::new(5, 6);
19.     assert_eq!(x, Tuple(5, 6));
20. }

```

12-58 new
 src/lib.rs 12-58
 12-58 src/lib.rs


```

1. extern crate proc_macro;
2. use {
3.     syn::{Token, DeriveInput, parse_macro_input},
4.     quote::*,
5.     proc_macro2,
6.     self::proc_macro::TokenStream,
7. };
8. #[proc_macro_derive(New)]
9. pub fn derive(input: TokenStream) -> TokenStream {
10.     let ast = parse_macro_input!(input as DeriveInput);
11.     let result = match ast.data {
12.         syn::Data::Struct(ref s) => new_for_struct(&ast, &s.fields),
13.         _ => panic!("doesn't work with unions yet"),
14.     };
15.     result.into()
16. }

```

12-58 1 7 TokenStream

8 [proc_macro_derive(New)]
[derive(New)]

9 derive pub
TokenStream

· parse_macro_input input syn DeriveInput
10

· ast.data syn Data syn
Struct DataStruct Enum DataEnum Union
DataUnion
new_for_struct 11 14

· result proc_macro2 TokenStream
into TokenStream 16
proc_macro2 proc_macro Rust 1.15
Rust 1.30 Rust

new_for_struct syn DeriveInput
proc_macro_derive 12-59

12-59 syn DeriveInput

```
1. // syn::DeriveInput
2. pub struct DeriveInput {
3.     pub attrs: Vec<Attribute>,
4.     pub vis: Visibility,
5.     pub ident: Ident,
6.     pub generics: Generics,
7.     pub data: Data,
8. }
```

12-59 DeriveInput

- **attrs** Vec syn Attribute syn Attribute
[repr C] Vec T
- **vis** syn Visibility
- **ident** syn Ident
- **generics** syn Generics
- **data** syn Data

DeriveInput trait 12-60

12-60 syn DeriveInput Parse

```
1. impl Parse for DeriveInput{...}
2. pub trait Parse: Sized {
3.     fn parse(input: ParseStream) -> Result<Self>;
4. }
```

syn 0.15 Parse Synom syn 0.15 Synom

Parse parse syn parse ParseStream
syn token TokenStream

parse_macro_input Parse
DeriveInput
parse_macro_input as as

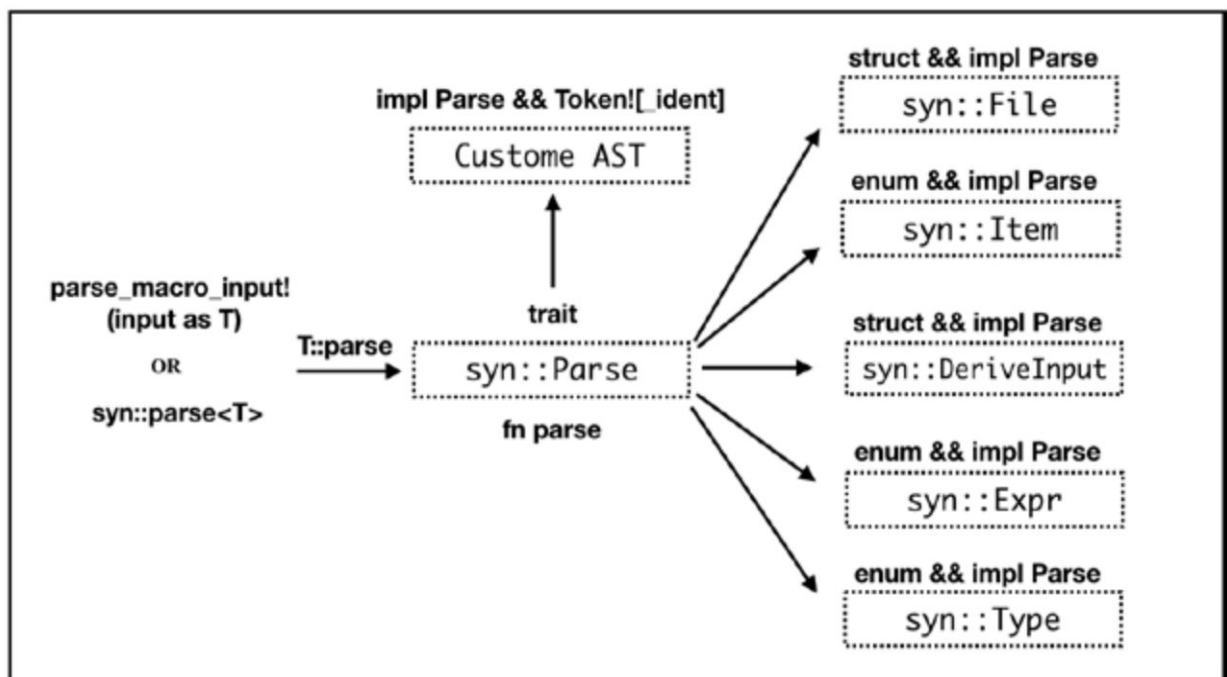
12-61

12-61

```
pub fn parse<T: Parse>(tokens: TokenStream) -> Result<T, Error>
```

12-58

Rust



12-11

12-62

12-62

```

1. // 其他代码同上，此处省略
2. fn new_for_struct(ast: &syn::DeriveInput, fields: &syn::Fields)
3.     -> proc_macro2::TokenStream
4. {
5.     match *fields {
6.         syn::Fields::Named(ref fields) => {
7.             new_impl(&ast, Some(&fields.named), true)
8.         },
9.         syn::Fields::Unit => {
10.            new_impl(&ast, None, false)
11.        },
12.        syn::Fields::Unnamed(ref fields) => {
13.            new_impl(&ast, Some(&fields.unnamed), false)
14.        },
15.    }
16. }

```

12-62 new_for_struct ast fields
 ast syn::DeriveInput fields syn::Fields
 proc_macro2::TokenStream derive
 proc_macro2::TokenStream

syn::Fields syn::Expr
 Fields syn::Fields::Named syn::Fields::Unit
 syn::Fields::Unnamed
 12-63

match fields new_impl
 ast

new_impl 12-63

12-63 src/lib.rs new_impl

```

1. fn new_impl(ast: &syn::DeriveInput,
2.     fields: Option<&syn::punctuated::Punctuated
3.         <syn::Field, Token![,]>
4.     >,
5.     named: bool) -> proc_macro2::TokenStream
6. {
7.     let struct_name = &ast.ident;
8.     let (impl_generics, ty_generics, where_clause) =
9.         ast.generics.split_for_impl();
10.    let (mut new, doc) = (
11.        syn::Ident::new("new", proc_macro2::Span::call_site()),
12.        format!("Constructs a new `{}`.", struct_name)
13.    );
14.    let args = "TODO";
15.    let inits = "TODO";
16.    quote! {
17.        impl #impl_generics #struct_name #ty_generics #where_clause {
18.            #[doc = #doc]
19.            pub fn #new(#(#args),*) -> Self {
20.                #struct_name #inits
21.            }
22.        }
23.    }
24. }

```

12-63 new_impl ast named fields Option &syn punctuated Punctuated syn Field Token[,] “”

· Option T fields fields None

· syn punctuated Punctuated T P

> Punctuated Field Token[,]

args inits
12-65

12-65 **src/lib.rs** **FieldExt** **a**

```
1. // 其他代码同上
2. struct FieldExt<'a> {
3.     ty: &'a syn::Type,
4.     ident: syn::Ident,
5.     named: bool,
6. }
```

12-65 **FieldExt** **a** **ty**
ident **named**
new

FieldExt **a** 12-66

12-66 **src/lib.rs** **FieldExt** **a**

```
1. impl<'a> FieldExt<'a> {
2.     pub fn new(field: &'a syn::Field, idx: usize, named: bool)
3.         -> FieldExt<'a> {
4.         FieldExt {
5.             ty: &field.ty,
6.             ident: if named {
7.                 field.ident.clone().unwrap()
8.             } else {
9.                 syn::Ident::new(
10.                     &format!("f{}", idx),
```

```

11.             proc_macro2::Span::call_site()
12.         )
13.     },
14.     named: named,
15. }
16. }
17. pub fn as_arg(&self) -> proc_macro2::TokenStream {
18.     let f_name = &self.ident;
19.     let ty = &self.ty;
20.     quote!(#f_name: #ty)
21. }
22. pub fn as_init(&self) -> proc_macro2::TokenStream {
23.     let f_name = &self.ident;
24.     let init = quote!(#f_name);
25.     if self.named {
26.         quote!(#f_name: #init)
27.     } else {
28.         quote!(#init)
29.     }
30. }
31. }

```

12-66 FieldExt a new
 as_arg proc_macro2 TokenStream
 as_init proc_macro2 TokenStream

215 new field syn Field
 ty Option Ident
 syn Fields idx

1721 as_arg FieldExt a ty
 ident quote

2230 as_init "f_name
 init" "init"

new 12-64

new_impl 12-67

12-67 src/lib.rs new_impl

```
1. fn new_impl(ast: &syn::DeriveInput,
2.     fields: Option<
3.         &syn::punctuated::Punctuated<syn::Field, Token![,]>
4.     >,
5.     named: bool) -> proc_macro2::TokenStream
6. {
7.     let struct_name = &ast.ident;
8.     let unit = fields.is_none();
9.     let empty = Default::default();
10.    let fields: Vec<_> = fields.unwrap_or(&empty).iter()
11.        .enumerate()
12.        .map(|(i, f)| FieldExt::new(f, i, named)).collect();
13.    let args = fields.iter().map(|f| f.as_arg());
14.    let inits = fields.iter().map(|f| f.as_init());
15.    let inits = if unit {
16.        quote!()
17.    } else if named {
18.        quote! [ { #(#inits),* } ]
19.    } else {
20.        quote! [ ( #(#inits),* ) ]
21.    };
22.    // 同上
23. }
```

12-67 8 21

8 unit 9 empty
Default default

10 12 fields FieldExt a

13 14 as_args as_init

15 21 quote
“{inits*}” “{arg1 arg1 arg2
arg2}” “inits*”
“arg1 arg2”

derive-new crate cargo test 12-68

12-68 [derive New] [new value=xxx]

```
1. #[derive(New)]
2. pub struct Fred {
3.     #[new(value = "1 + 2")]
4.     pub x: i32,
5.     pub y: String,
6.     #[new(value = "vec![-42, 42]")]
7.     pub z: Vec<i8>,
8. }
9. #[test]
10. fn test_struct_with_values() {
11.     let x = Fred::new("Fred".to_owned());
12.     assert_eq!(
13.         x,
14.         Fred { x: 3, y: "Fred".to_owned(), z: vec![-42, 42] }
15.     );
16. }
```

12-68 [proc_macro_derive New attributes new] derive attributes

12.3

Rust Nightly Rust [feature plugin_registrar]

librustc_plugin

· register_syntax_extension

· **register_custom_derive** □ □ □
register_syntax_extension□□□□□□□□□□□□□□□□

· **register_macro** □□□□register_syntax_extension□□□□□
□□□Bang□□

· register_attribute□□□□□□□□□□
□□□□ □lint□□□llvm□□□
□□□□□□□□□□□□□□□□ cargo new □□□□□□ lib □□□□□
plugin_demo□□□□tests/test.rs□□□□□□□□□□□□□□□□12-69□□□
□□□□12-69□**plugin_demo**□□□□

```
├─ Cargo.toml
├─ src
│   └─ lib.rs
└─ tests
    └─ test.rs
```

□Cargo.toml□□□□lib□□□□plugin□□□□□□□□□□12-70□□□
□□□□12-70□**Cargo.toml**□□□□**lib**□□□□**plugin**□□

```
[lib]
plugin = true
```

□□tests/test.rs□□□□□□□□□□□□□□□□12-71□□□
□□□□12-71□□**tests/test.rs**□□□□□□□□

```
1.  #![feature(plugin)]
2.  #![plugin(plugin_demo)]
3.  #[test]
4.  fn test_plugin() {
5.      assert_eq!(roman_to_digit!(MMXVIII), 2018);
6.  }
```

□□□□12-71□□□□□□□□**[feature□plugin□]** □□□□□□1□□□□□□□□□□
Nightly□□□□Rust□□□□

□□□□2□□□□□□□□**[plugin□plugin_demo□]** □□□□plugin_demo□□□□
□□□□□□□□□□

4 roman_to_digit MMXVIII2018

src/lib.rs12-72

12-72src/lib.rs

```
1.  #![feature(plugin_registrar, rustc_private)]
2.  extern crate syntax;
3.  extern crate rustc;
4.  extern crate rustc_plugin;
5.  use self::syntax::parse::token;
6.  use self::syntax::tokenstream::TokenTree;
7.  use self::syntax::ext::base::{ExtCtxt, MacResult, DummyResult,
    MacEager};
8.  use self::syntax::ext::build::AstBuilder;
9.  use self::syntax::ext::quote::rt::Span;
10. use self::rustc_plugin::Registry;
11. static ROMAN_NUMERALS: &'static [(&'static str, usize)] = &[
12.     ("M", 1000), ("CM", 900), ("D", 500), ("CD", 400),
13.     ("C", 100), ("XC", 90), ("L", 50), ("XL", 40),
14.     ("X", 10), ("IX", 9), ("V", 5), ("IV", 4),
15.     ("I", 1)
16. ];
```

12-72 1 [feature plugin_registrar rustc_private] plugin_registrar rustc_private Nightly

210 Syntax Rust libsyntax rustc_plugin Rust librustc_plugin

1116 ROMAN_NUMERALS

12-73

12-73src/lib.rs expand_roman

```

1. fn expand_roman(cx: &mut ExtCtxt, sp: Span, args: &[TokenTree])
2.     -> Box<MacResult + 'static>
3. {
4.     let text = match args[0] {
5.         TokenTree::Token(_, token::Ident(s, _)) => s.to_string(),
6.         _ => {
7.             cx.span_err(sp, "argument should be a single identifier");
8.             return DummyResult::any(sp);
9.         }
10.    };
11.    let mut text = &*text;
12.    let mut total = 0;
13.    while !text.is_empty() {
14.        match ROMAN_NUMERALS
15.            .iter().find(|&(rn, _)| text.starts_with(rn))
16.        {
17.            Some(&(rn, val)) => {
18.                total += val;
19.                text = &text[rn.len()..];
20.            }
21.            None => {
22.                cx.span_err(sp, "invalid Roman numeral");
23.                return DummyResult::any(sp);
24.            }
25.        }
26.    }
27.    MacEager::expr(cx.expr_usize(sp, total))
28. }

```

12-73 expand_roman

• **cx** ExtCtxt

• **Span**

• **TokenTree**

```
pub struct BoxMacResult+ {static trait AST
MacResult trait trait AST
AST
```

```
4 10 text args[0] TokenTree
match token Ident s text
roman_to_digit MMXVIII
cx.span_err sp DummyResult any
sp
```

```
12 total 0
13 26 text
ROMAN_NUMERALS
total ROMAN_NUMERALS
cx.span_err sp sp
```

```
27 MacEager expr total sp cx.expr_size
total usize
```

```
MacEager Rust expr
pat items impl_items stmts ty
```

```
roman_to_digit 12-74
12-74 src/lib.rs roman_to_digit
1. #[plugin_registrar]
2. pub fn roman_to_digit(reg: &mut Registry) {
3.     reg.register_macro("roman_to_digit", expand_roman);
4. }
```

```
12-74 1 [plugin_registrar]
roman_to_digit reg register_macro
roman_to_digit expand_roman
```

```
plugin_demo cargo rest
Rust 1.30
```

Rustのlibsyntaxライブラリはlibsyntaxライブラリを再実装したライブラリです。

このライブラリはASTとTokenStreamのインターフェイスを提供します。2.0バージョンでは、以前のバージョンと互換性のあるAPIを提供しています。

12.4 関数

このセクションでは、RustのASTとTokenStreamのインターフェイスを定義する関数について説明します。

RustのASTとTokenStreamのインターフェイスを定義する関数。

この関数はRustのASTとTokenStreamのインターフェイスを定義するBang関数です。Bang関数はmacro_rulesとRust 2018のASTとTokenStreamのインターフェイスを定義するmacro関数です。

RustのASTとTokenStreamのインターフェイスを定義するAST関数です。AST関数はRustのASTとTokenStreamのインターフェイスを定義するRust関数です。Rust関数はRustのASTとTokenStreamのインターフェイスを定義するTokenStream関数です。

この関数はBang関数です。Bang関数はRust 1.1のASTとTokenStreamのインターフェイスを定義するBang関数です。Rust 1.30のASTとTokenStreamのインターフェイスを定義する[feature=proc_macro]と[feature=custom_attribute]のASTとTokenStreamのインターフェイスを定義するBang関数です。

この関数はsynのquote関数です。synのquote関数はRustのASTとTokenStreamのインターフェイスを定義するBang関数です。RustのASTとTokenStreamのインターフェイスを定義するBang関数はRustのASTとTokenStreamのインターフェイスを定義するRust関数です。

この関数はRustのASTとTokenStreamのインターフェイスを定義するRust関数です。RustのASTとTokenStreamのインターフェイスを定義するRust関数はRustのASTとTokenStreamのインターフェイスを定義するRust関数です。

Web `rocket` `Rust 0.3` `12-75`

`12-75 rocket`

```
1.  #![feature(plugin, decl_macro)]
2.  #![plugin(rocket_codegen)]
3.  extern crate rocket;
4.  #[get("/")]
5.  fn hello() -> &'static str {
6.      "Hello, world!"
7.  }
8.  fn main() {
9.      rocket::ignite().mount("/", routes![hello]).launch();
10. }
```

`12-75` Web `rocket` Hello World `1` `plugin` `decl_macro` `rocket` `macro`

`2` `[plugin rocket_codegen]` `rocket_codegen`

`4` `[get /]` `5` `hello` `"GET https://domain/"` `HTTP` `hello`

`Rust` `Rust` `"`

[1] [rocket_codegen](#)

[2] `Rust` `src/libsyntax/ext/tt/macro_rules.rs`

[3] [rocket](#)

[4] `TDD`

[5] [GitHub nrc/derive-new](#)

第13章 内存管理

内存管理是操作系统中最重要的一部分。

在操作系统中，内存管理的主要任务是分配和回收内存。操作系统通过维护一个内存池，来跟踪系统中所有内存的使用情况。当用户程序请求内存时，操作系统会从内存池中分配出一块内存。当用户程序不再需要这块内存时，操作系统会将其回收，并放回内存池中，以便将来再次使用。

Rust 内存管理的主要特点是安全。Rust 通过编译时检查，确保程序不会发生内存泄漏或越界访问。C 语言在 UNIX、Linux 和 Windows 上都有实现，但 Rust 则是一个跨平台的语言。Rust 的内存管理模型与 C 语言不同，它不需要程序员手动管理内存。

Rust 内存管理的主要特点是安全。Rust 通过编译时检查，确保程序不会发生内存泄漏或越界访问。C 语言在 UNIX、Linux 和 Windows 上都有实现，但 Rust 则是一个跨平台的语言。Rust 的内存管理模型与 C 语言不同，它不需要程序员手动管理内存。

Rust 内存管理的主要特点是安全。Rust 通过编译时检查，确保程序不会发生内存泄漏或越界访问。C 语言在 UNIX、Linux 和 Windows 上都有实现，但 Rust 则是一个跨平台的语言。Rust 的内存管理模型与 C 语言不同，它不需要程序员手动管理内存。

Safe Rust 内存管理的主要特点是安全。Safe Rust 通过编译时检查，确保程序不会发生内存泄漏或越界访问。Unsafe Rust 则是一个跨平台的语言。Unsafe Rust 的内存管理模型与 Safe Rust 不同，它需要程序员手动管理内存。

13.1 Unsafe Rust

Unsafe Rust 内存管理的主要特点是安全。Unsafe Rust 通过编译时检查，确保程序不会发生内存泄漏或越界访问。Safe Rust 则是一个跨平台的语言。Safe Rust 的内存管理模型与 Unsafe Rust 不同，它需要程序员手动管理内存。

本章 13-1 节 unsafe 内存管理的主要特点是安全。

```

1. fn main(){
2.     unsafe {
3.         let mut a = "hello";
4.         let b = &a;
5.         let c = &mut a;
6.     }
7. }

```

13-1 unsafe a Rust Unsafe Rust Safe Rust

Unsafe Rust

-
- unsafe
-
- unsafe trait
- **Union**

Rust “” Rust

Unsafe Rust Safe Rust

- Unsafe Rust
- Unsafe Rust
- Unsafe Rust

Unsafe Rust Safe Rust Rust

13.1.1 Unsafe

unsafe unsafe Unsafe Rust

- **unsafe** trait

· **unsafe** 与 `Unsafe Rust` 的区别

unsafe 是什么

Rust 中 `unsafe` 与 `trait` 和 `String` 的关系
13-2

13-2 `String` 与 `unsafe`

```
1. pub unsafe fn from_utf8_unchecked(bytes: Vec<u8>) -> String {  
2.     String { vec: bytes }  
3. }
```

13-2 `String` 与 `unsafe` 的 `from_utf8_unchecked`
与 `unsafe` 和 `Vec<u8>` 的关系
`String`

与 `String` 和 `Unsafe Rust`
与 `Safe Rust`
`unsafe` 与 `bytes`
UTF-8
“”

`from_utf8_unchecked` 与 “” UTF-8
`unsafe` 与 `unsafe`
“” “” “”
`unsafe`

Rust 与 “”
`unsafe`
“” `unsafe`
Bug “”

`unsafe` 与 `trait`

`unsafe trait` `Send` `Sync` 与 Rust
`trait` Rust
`unsafe` `Send` `Sync`

`unsafe trait` `std` `str` `pattern` `Searcher`
13-3

13-3 Searcher

```
1. pub unsafe trait Searcher<'a> {
2.     fn next(&mut self) -> SearchStep;
3.     // ...
4. }
```

```

    13-3Searcherunsafe traitunsafe
    traitnext

```

```

    Searcher next UTF-8
    Searcher next
    Searcher
    Searcher
    trait unsafe trait
    trait unsafe trait
    trait unsafe trait unsafe
impl

```

unsafe 

□ unsafe □□□□□□□□□□□□□□ unsafe □□□□□□□□□□ 13-4□
□□

13-4 unsafe

```
1. fn main() {
2.     let hello = vec![104, 101, 108, 108, 111];
3.     let hello = unsafe {
4.         String::from_utf8_unchecked(hello)
5.     };
6.     assert_eq!("hello", hello);
7. }
```

```

000013-4000from_utf8_unchecked00000000000000000000
unsafe000000000013-50000000

```

13-5 unsafe unsafe

[illegible]

13-5 unsafe unsafe
unsafe unsafe
unsafe

13.1.2

Rust
Rust

unsafe
13-6

13-6 unsafe

```
1. static mut COUNTER: u32 = 0;
2. fn main() {
3.     let inc = 3;
4.     unsafe {
5.         COUNTER += inc;
6.         println!("COUNTER: {}", COUNTER);
7.     }
8. }
```

13-6 COUNTER unsafe
unsafe unsafe
C
C

13.1.3 Union

Rust C Union Union Enum Enum
Tagged Union Tag Tag
Union Tag
Enum

Union Enum
Rust Union Rust C

“ ” 13-7

13-7 Union Struct Enum

```
1. #[repr(C)]
2. union U {
3.     i: i32,
4.     f: f32,
5. }
6. #[repr(C)]
7. struct Value{
8.     tag: u8,
9.     value: U,
10. }
11. #[repr(C)]
12. union MyZero {
13.     i: Value,
14.     f: Value,
15. }
16. enum MyEnumZero {
17.     I(i32),
18.     F(f32),
19. }
20. fn main(){
21.     let int_0 = MyZero{i: Value{tag: b'0', value: U { i: 0 } } };
22.     let float_0 = MyZero{i: Value{tag: b'1', value: U { f: 0.0 } } };
23. }
```

13-7 Union Struct Enum MyZero
0 0.0

1 5 union Union i f i32 f32
Union **repr C**
Rust C [repr C]

6 10 Value tag value Enum
Enum tag Value [repr C]

value 0 U

11 15 MyZero if Value
16 19 Enum MyEnumZero

main MyZero 13-7
 Rust **Union** **Copy** Non-
Copy MyZero Copy **[feature**
untagged_unions]

13-7 13-8

13-8 **13-7**

```

1.  #[repr(u32)]
2.  enum Tag { I, F }
3.  #[repr(C)]
4.  union U {
5.      i: i32,
6.      f: f32,
7.  }
8.  #[repr(C)]
9.  struct Value {
10.     tag: Tag,
11.     u: U,
12. }
13. fn is_zero(v: Value) -> bool {
14.     unsafe {
15.         match v {
16.             Value { tag: Tag::I, u: U { i: 0 } } => true,
17.             Value { tag: Tag::F, u: U { f: 0.0 } } => true,
18.             _ => false,
19.         }
20.     }
21. }
22. fn main() {
23.     let int_0 = Value{tag: Tag::I, u: U{i: 0}};
24.     let float_0 = Value{tag: Tag::F, u: U{f: 0.0}};
25.     assert_eq!(true, is_zero(int_0));
26.     assert_eq!(true, is_zero(float_0));
27.     assert_eq!(4, std::mem::size_of::<U>());
28. }

```

1 2 Enum Tag I F
 [repr u32] Rust

3 7 union U i f 13-7

8 12 Value tag u Tag U
 13-7

5. `r1` `r2` 的乘积
 6. `address` 的 `as` 操作
 7. `r3`
 8. `r1` `r2` 的乘积
 9. `s`
 10. `unsafe`
 11. `Segmentation Fault`
 12. `r3`

13.2 Unsafe

1. `unsafe`
 2. `unsafe`
 3. `unsafe`
 4. `unsafe`
 5. `Unsafe Rust`

13.2.1

1. `Unsafe Rust`
 2. `Rust`
 3. `C`
 4. `std::ptr::null`
 5. `is_null`
 6. `offset`
 7. `read/write`
 8. `replace/swap`
 9. `13-11`
 10. `13-11`

```

1. fn main() {
2.     let p: *const u8 = std::ptr::null();
3.     assert!(p.is_null());
4.     let s: &str = "hello";
5.     let ptr: *const u8 = s.as_ptr();
6.     assert!(!ptr.is_null());
7.     let mut s = [1, 2, 3];
8.     let ptr: *mut u32 = s.as_mut_ptr();
9.     assert!(!ptr.is_null());
10. }

```

13-11 `std::ptr::null()` 返回一个 `is_null` 为 `true` 的指针

4 6 `&str` `s.as_ptr()` 返回一个 `is_null` 为 `false` 的指针 `*const u8` 表示一个指向常量的无符号字节指针

7 9 `s.as_mut_ptr()` 返回一个 `*mut u32` 类型的指针，指向一个可变的 `u32` 值

使用 `unsafe` 块

`offset`

`offset` 方法返回一个指向字符串中指定偏移量的指针。偏移量从 0 开始，按字节递增。例如，偏移量为 1 指向第一个字符，偏移量为 3 指向第四个字符，偏移量为 255 指向字符串末尾（如果字符串长度小于 255，则指向字符串末尾）。13-12

13-12 `offset`

```

1. fn main() {
2.     let s: &str = "Rust";
3.     let ptr: *const u8 = s.as_ptr();
4.     unsafe {
5.         println!("{:?}", *ptr.offset(1) as char); // u
6.         println!("{:?}", *ptr.offset(3) as char); // t
7.         println!("{:?}", *ptr.offset(255) as char); // ÿ
8.     }
9. }

```

13-12 2 3 ***const u8** ptr
s offset s

4 8 offset unsafe unsafe
***ptr.offset(1) as char** ***ptr.offset(1) as char**
as offset
5 6 u t

offset
unsafe 7

read/write

read write unsafe
13-13 read

13-13 read/write

```

1. fn main() {
2.     let x = "hello".to_string();
3.     let y: *const u8 = x.as_ptr();
4.     unsafe {
5.         assert_eq!(y.read() as char, 'h');
6.     }
7.     let x = [0, 1, 2, 3];
8.     let y = x[0..].as_ptr() as *const [u32; 4];
9.     unsafe {
10.        assert_eq!(y.read(), [0,1,2,3]);
11.    }
12.    let x = vec![0, 1, 2, 3];
13.    let y = &x as *const Vec<i32>;
14.    unsafe {
15.        assert_eq!(y.read(), [0,1,2,3]);
16.    }
17.    let mut x = "";
18.    let y = &mut x as *mut &str;
19.    let z = "hello";
20.    unsafe {
21.        y.write(z);
22.        assert_eq!(y.read(), "hello");
23.    }
24. }

```

13-13 2 String x 3 x **as_ptr**
 String *const
u8 4 6 unsafe y read

read unsafe read

7 11 x as_ptr *const
[u32 4] y y read
***const [u32 3]** read

1216xas_ptr
xasread

as_ptras_ptr

read1723writewrite
writeunsafe

replace/swap

replaceswap13-14

13-14replace/swap

```

1. fn main() {
2.     let mut v: Vec<i32> = vec![1, 2];
3.     let v_ptr : *mut i32 = v.as_mut_ptr();
4.     unsafe{
5.         let old_v = v_ptr.replace(5);
6.         assert_eq!(1, old_v);
7.         assert_eq!([5, 2], &v[..]);
8.     }
9.     let mut v: Vec<i32> = vec![1, 2];
10.    let v_ptr = &mut v as *mut Vec<i32>;
11.    unsafe{
12.        let old_v = v_ptr.replace(vec![3,4,5]);
13.        assert_eq!([1, 2], &old_v[..]);
14.        assert_eq!([3, 4, 5], &v[..]);
15.    }
16.    let mut array = [0, 1, 2, 3];
17.    let x = array[0..].as_mut_ptr() as *mut [u32; 2];
18.    let y = array[1..].as_mut_ptr() as *mut [u32; 2];
19.    unsafe {
20.        assert_eq!([0, 1], x.read());
21.        assert_eq!([1, 2], y.read());
22.        x.swap(y);
23.        assert_eq!([1, 0, 1, 3], array);
24.    }
25. }

```

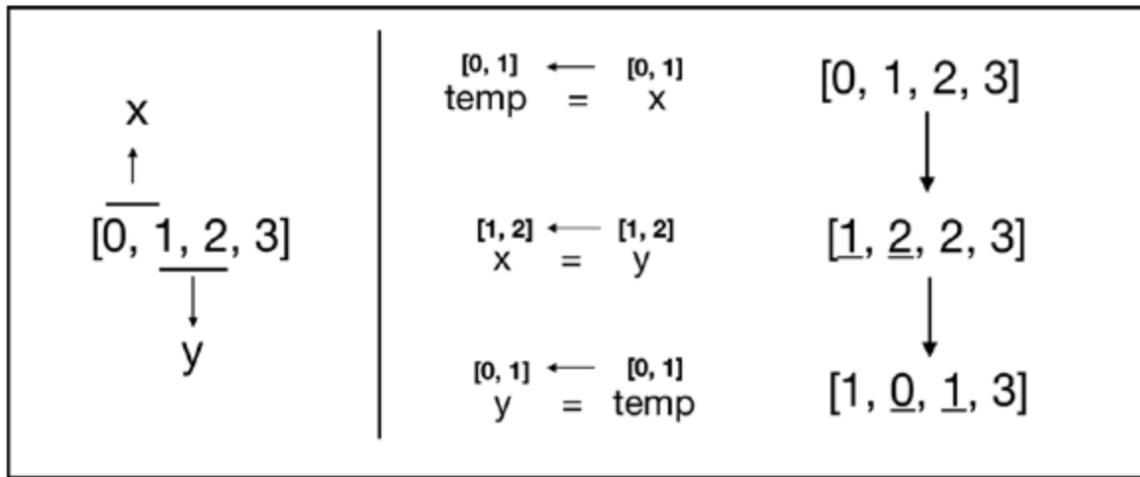
13-14 2 3 as_mut_ptr v
 4 8 unsafe replace v
 5 old_v 1 v [5 2]

9 15 v
 unsafe replace Vec i32

16 24 swap

16 18 array as_mut_ptr x
y *mut [u32; 2]

19 24 read x y [0; 1] [1; 2] x
swap y [1; 0; 1; 3] 13-1



13-1 swap x y

13-1 swap 13-1 unsafe

std mem swap fn swap T
x &mut T y &mut T
replace

Unsafe Rust Vec T
insert Safe Rust insert
&mut Vec T Safe Rust

13-15 Vec T insert

13-15 Vec T insert

```

1. pub fn insert(&mut self, index: usize, element: T) {
2.     let len = self.len();
3.     assert!(index <= len);
4.     if len == self.buf.cap() {
5.         self.reserve(1);
6.     }
7.     unsafe {
8.         {
9.             let p = self.as_mut_ptr().offset(index as isize);
10.            ptr::copy(p, p.offset(1), len - index);
11.            ptr::write(p, element);
12.        }
13.        self.set_len(len + 1);
14.    }
15. }

```

代码13-15 实现 insert 函数。函数参数为 &mut self、index 和 element。其中 self 为 Vec<T> 的引用，index 为插入位置，element 为要插入的元素。

代码2和3 检查 index 是否大于 len。如果大于，则返回错误。如果小于，则继续执行。

代码4和6 检查是否需要 reserve。如果 len 等于 capacity，则需要 reserve。reserve 函数会返回一个 usize 类型的值，表示需要 reserve 的容量。

代码7 使用 unsafe 块。

代码8和12 使用 as_mut_ptr 函数获取指向元素的指针。然后使用 offset 函数将指针移动到 index 位置。代码9和10 使用 ptr::copy 函数将元素复制到 index 位置。代码11 使用 ptr::write 函数将 element 写入到 index 位置。

代码13 调用 self.set_len 函数，将 len 增加 1。

代码14 使用 unsafe 块。代码15 返回。insert 函数返回一个 Vec<T> 类型的值，表示插入后的 Vec。insert 函数返回的 Vec 与 self 指向的 Vec 是同一个 Vec。

13.2.2 插入元素

`subtype` `supertype` **A** **B** A B

subtype polymorphism

Ring **Circle**

Liskov Substitution Principle **LSP**

□ □ □ □ □ □ □

```

variance Cat Animal Cat
Animal List Cat List Animal

```

□ □ □ □ □ □ □ □ □ □ □ □

```
· covariant[[Cat,Animal]]List[Cat]List[Animal]
```

```
·  contravariant Cat Animal List
Animal List Cat
```

```
· invariant [] [] Cat Animal []  
[] List Animal [] List Cat [] []
```

```
Rust
// "long short"
long short
short
static str & a
str
```

```

000000000000000000000000000000000000 Unsafe00000000
00000000000000000000000000000000

```

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

□□□□13-16□□□□□□□□□□□□□□□□

13-16 MyCell T

```

1. struct MyCell<T> {
2.     value: T,
3. }
4. impl<T: Copy> MyCell<T> {
5.     fn new(x: T) -> MyCell<T> {
6.         MyCell { value: x }
7.     }
8.     fn get(&self) -> T {
9.         self.value
10.    }
11.    fn set(&self, value: T) {
12.        use std::ptr;
13.        unsafe {
14.            ptr::write(&self.value as *const _ as *mut _, value);
15.        }
16.    }
17. }

```

13-16 1 3 MyCell T value
 4 17 MyCell T new get set new get
 set

11 16 set unsafe ptr write
 ptr write &self.value
 Rust

MyCell T 13-17

13-17 MyCell T

```

1. fn step1<'a>(r_c1: &MyCell<&'a i32>) {
2.     let val: i32 = 13;
3.     step2(&val, r_c1);
4.     println!("step1 value: {}", r_c1.value);
5. }
6. fn step2<'b>(r_val: &'b i32, r_c2: &MyCell<&'b i32>) {
7.     r_c2.set(r_val);
8. }

```

```

9.  static X: i32 = 10;
10. fn main() {
11.     let cell = MyCell::new(&X);
12.     step1(&cell);
13.     println!(" end value: {}", cell.value);
14. }

```

13-17 15 step1 r_c1 &MyCell &a i32 val &val step2

6 8 step2 &b i32 &MyCell &a i32

9 X

main 11 &X MyCell cell 12 &cell step1 cell.value

13-17 3 step1 step2 val &val step2 set &val &val step2 step1 step1 val &val cell.value

Rust

MyCell T Rust 13-17 X &X static main step1 &MyCell &static i32 step1 &MyCell &a i32 MyCell T &static i32 &a i32 &MyCell &static i32 &MyCell &a i32

Rust " 3 step1 &val a static

MyCell T

PhantomData<T>

PhantomData<T> 是 std::marker 模块中的一个 trait，它用于告诉编译器“这个值在编译时是已知的，但在运行时是不可知的”。PhantomData<T> 通常用于实现不可变引用（immutable references）。

- 它实现了 Copy 和 Clone trait。
- 它实现了 Drop trait。
- 它实现了 Send 和 Sync trait。

PhantomData<T> 在 Rust 13-17 版本中被引入。在 13-18 版本中，它被用于实现不可变引用。

13-18 版本的 PhantomData<T> 和 MyCell<T>

```
1. use std::marker::PhantomData;
2. struct MyCell<T> {
3.     value: T,
4.     mark: PhantomData<fn(T)> ,
5. }
6. impl<T: Copy> MyCell<T> {
7.     fn new(x: T) -> MyCell<T> {
8.         MyCell { value: x , mark: PhantomData}
9.     }
10.    fn get(&self) -> T {
11.        self.value
12.    }
13.    fn set(&self, value: T) {
14.        use std::ptr;
15.        unsafe {
16.            ptr::write(&self.value as *const _ as *mut _, value);
17.        }
18.    }
19. }
```

在 13-18 版本中，MyCell<T> 是一个不可变的引用。它使用 PhantomData<fn(T)> 来告诉编译器，这个值在编译时是已知的，但在运行时是不可知的。在 Rust 13-18 版本中，Rust 编译器会检查这个 trait 是否被正确实现。

MyCellT13-17

error[E0597]: `val` does not live long enough

--> src/main.rs:

```
|     step2(&val, r_cl);  
|           ^^^ borrowed value does not live long enough  
|     println!("step1 value: {}", r_cl.value);  
| }  
| - borrowed value only lives until here
```

Rust

Rust

·&a T a T *const T

·&a mut T a T

·Fn T-U T U

·Box T Vec T

·UnsafeCell T Cell T RefCell T Mutex T

T *mut T

&mut&static str &mut&a str

13-16

UnsafeCell T *mut T

Phantomdata T

·PhantomData T T

·PhantomData &a T a T

·PhantomData &a mut T a T

·PhantomData *const T T

·PhantomData *mut T T

·PhantomData fn T T

·PhantomData fn T-T T T

·PhantomData fn T-T T T

· `fn T trait { static str &a str fn &a str fn &static str }`

· `&static str &str &str &static str`

13-20

13-20 fn T

```
1. fn foo(input: &str) {
2.     println!("{:?}", input);
3. }
4. fn bar(f: fn(&'static str), v: &'static str) {
5.     (f) (v);
6. }
7. fn main() {
8.     let v : &'static str = "hello";
9.     bar(foo, v);
10. }
```

13-20 `bar fn &static str` `main` 9 `foo` `bar` `foo fn &str fn &str fn &static str`

Rust

`Unsafe` `Unbound Lifetime`

13.2.3

`Unsafe` `Unbound Lifetime`

· `&*raw_ptr`

· `std mem transmute` `transmute &T &U foo`

13-21

13-21

```
1. fn foo<'a>(input: *const u32) -> &'a u32 {
2.     unsafe {
3.         return &*input
4.     }
5. }
6. fn main() {
7.     let x;
8.     {
9.         let y = 42;
10.        x = foo(&y);
11.    }
12.    println!("hello: {}", x);
13. }
```

13-21 foo input *const u32
return &*input "&*input"

main 8 11 foo &y
12 x Safe Rust foo y
y x Rust
foo
Rust

Debug "hello 42" Release
"hello 1151157120"

13-22

13-22 transmute

```

1. use std::mem::transmute;
2. fn main() {
3.     let x: &i32;
4.     {
5.         let a = 12;
6.         let ptr = &a as *const i32;
7.         x = unsafe { transmute::<*const i32, &i32>(ptr) };
8.     }
9.     println!("hello {}", x);
10. }

```

13-22 std::mem::transmute T U
T U unsafe

main ptr transmute &i32
x 13-21 Release
"hello-697728128"

13.2.4 Drop

Drop dropck

Safe Rust dropck

Safe Rust Rust 13-23

13-23 dropck

```

1. use std::fmt;
2. #[derive(Copy, Clone, Debug)]
3. enum State { InValid, Valid }
4. #[derive(Debug)]
5. struct Hello<T: fmt::Debug>(&'static str, T, State);
6. impl<T: fmt::Debug> Hello<T> {
7.     fn new(name: &'static str, t: T) -> Self {
8.         Hello(name, t, State::Valid)
9.     }
10. }
11. impl<T: fmt::Debug> Drop for Hello<T> {
12.     fn drop(&mut self) {
13.         println!("drop Hello({}, {:?}, {:?})",
14.             self.0,
15.             self.1,
16.             self.2);
17.         self.2 = State::InValid;
18.     }
19. }
20. struct WrapBox<T> {
21.     v: Box<T>,
22. }
23. impl<T> WrapBox<T> {
24.     fn new(t: T) -> Self {
25.         WrapBox { v: Box::new(t) }
26.     }
27. }
28. fn f1() {
29.     // let x; let y;
30.     let (x, y);
31.     x = Hello::new("x", 13);
32.     y = WrapBox::new(Hello::new("y", &x));
33. }
34. fn main() {
35.     f1();
36. }

```

13-23 2 3 State
4 19 Hello T new
Hello

20 27 WrapBox T Box T
WrapBox T new

28 33 f1 30
x y 32 WrapBox T x

13-24

13-24 13-23

error[E0597]: `x` does not live long enough

--> src/main.rs:

```
| y = WrapBox::new(Hello::new("y", &x));
```

```
|                                     ^ borrowed value does not live  
long enough
```

```
| }
```

```
| - `x` dropped here while still borrowed
```

= note: values in a scope are dropped in the opposite order they are created

13-24 x y x y
x y &x Safe Rust

x y 13-23 29
x y x y
13-25

13-25 13-23

```
drop Hello(y, Hello("x", 13, Valid), Valid)
```

```
drop Hello(x, 13, Valid)
```

13-25 y x
Valid

Safe Rust WrapBox T Box T Rust
WrapBox T Box T T WrapBox T Drop
Rust WrapBox T T

[may_dangle] dropck

13-23 `MyBox<T>` 的
Box<T> 的 nightly Rust
[feature=allocator_api] 13-26

13-26

```
1. #![feature(allocator_api)]
2. use std::alloc::{GlobalAlloc, System, Layout};
3. use std::ptr;
4. use std::mem;
5. // 此处省略 State、Hello、impl Hello 等定义
6. struct MyBox<T> {
7.     v: *const T,
8. }
9. impl<T> MyBox<T> {
10.     fn new(t: T) -> Self {
11.         unsafe {
12.             let p = System.alloc(Layout::array::<T>(1).unwrap());
13.             let p = p as *mut T;
14.             ptr::write(p, t);
15.             MyBox { v: p }
16.         }
```

```

17.     }
18. }
19. impl<T> Drop for MyBox<T> {
20.     fn drop(&mut self) {
21.         unsafe {
22.             let p = self.v as *mut _;
23.             System.dealloc(p,
24.                 Layout::array::<T>(mem::align_of::<T>()).unwrap());
25.         }
26.     }
27. }
28. fn f2() {
29.     {
30.         let (x1, y1);
31.         x1 = Hello::new("x1", 13);
32.         y1 = MyBox::new(Hello::new("y1", &x1));
33.     }
34.     {
35.         let (x2, y2);
36.         x2 = Hello::new("x2", 13);
37.         y2 = MyBox::new(Hello::new("y2", &x2));
38.     }
39. }
40. fn main() {
41.     // f1();
42.     f2();
43. }

```

13-26 13-23 State Hello Hello
new drop

1 2 [feature allocator_api] std alloc
GlobalAlloc System Layout
feature Rust Nightly

3 4 std ptr std mem

```

    68  MyBox<T> v{*const T}
    9 18  MyBox<T> new new
System.alloc LayoutarrayT1 T
    MyBox<T>
    1927  MyBox<T>Drop drop ptr read
    v T System.dealloc T
LayoutarrayTmemalign_ofT T

```

```

    2839  f2 main f2 f1
    13-26 13-27
    13-2713-26

error[E0597]: `x1` does not live long enough
--> src/main.rs:
|         y1 = MyBox::new(Hello::new("y1", &x1));
|                                     ^^ borrowed value does not live
long enough
|     }
|     - `x1` dropped here while still borrowed
= note: values in a scope are dropped in the opposite order they are created
error[E0597]: `x2` does not live long enough
--> src/main.rs:
|     y2 = MyBox::new(Hello::new("y2", &x2));
|                                     ^^ borrowed value does not live long
enough
|     }
|     - `x2` dropped here while still borrowed
= note: values in a scope are dropped in the opposite order they are created

```

```

    13-27
drop T 13-26 MyBox<T>
drop T
[may_dangle]
[may_dangle] drop 13-28

```


13-28 drop

```
1. #![feature(allocator_api, dropck_eyepatch)]
2. //其他同上
3. unsafe impl<#[may_dangle] T> Drop for MyBox<T> {
4.     fn drop(&mut self) {
5.         unsafe {
6.             println!("mybox drop");
7.             let p = self.v as *mut _;
8.             System.dealloc(p,
9.                 Layout::array::<T>(mem::align_of::<T>()).unwrap());
10.        }
11.    }
12. }
```

13-28 1 [feature allocator_api dropck_eyepatch]

3 “unsafe impl [may_dangle] T Drop for MyBox T ” “ [may_dangle] T ” drop many_dangle may dangle pointer unsafe impl

drop T 13-29

13-29 drop f2

```
1. // 其他代码同上
2. unsafe impl<#[may_dangle] T> Drop for MyBox<T> {
```

```

3.     fn drop(&mut self) {
4.         unsafe {
5.             ptr::read(self.v); // 此处新增
6.             let p = self.v as *mut _;
7.             System.dealloc(p,
8.                 Layout::array::<T>(mem::align_of::<T>()).unwrap());
9.         }
10.    }
11. }
12. fn f2() {
13.     {
14.         let (x1, y1);
15.         x1 = Hello::new("x1", 13);
16.         y1 = MyBox::new(Hello::new("y1", &x1));
17.     }
18.     {
19.         let (y2, x2); // 此处改变
20.         x2 = Hello::new("x2", 13);
21.         y2 = MyBox::new(Hello::new("y2", &x2));
22.     }
23. }

```

13-29 5 19
 5 ptr read T MyBox T drop
 T

19 x2 y2 13-30

13-30

```

drop Hello(y1, Hello("x1", 13, Valid), Valid)
drop Hello(x1, 13, Valid)
drop Hello(x2, 13, Valid)
drop Hello(y2, Hello("x2", 13, Invalid), Valid)

```

13-30 `x1` `y1` `x2` `y2`
Hello **InValid** `T`

Rust Rust drop

PhantomData `T` **drop**

`MyBox` `T` Rust
`MyBox` `T` `T` drop `T`
`MyBox` `T` `MyBox` `T` drop `T`
PhantomData `T`

13-29 `MyBox2` `T` 13-31
13-31 `MyBox2` `T`

```

1. // 其余代码同上
2. use std::marker::PhantomData;
3. struct MyBox2<T> {
4.     v: *const T,
5.     _pd: PhantomData<T>,
6. }
7. impl<T> MyBox2<T> {
8.     fn new(t: T) -> Self {
9.         unsafe{
10.             let p = System.alloc(Layout::array::<T>(1).unwrap());
11.             let p = p as *mut T;
12.             ptr::write(p, t);
13.             MyBox2 { v: p, _pd: Default::default() }
14.         }
15.     }
16. }
17. unsafe impl<#[may_dangle] T> Drop for MyBox2<T> {
18.     fn drop(&mut self) {
19.         unsafe {
20.             ptr::read(self.v);
21.             let p = self.v as *mut _;
22.             System.dealloc(p,
23.                 Layout::array::<T>(mem::align_of::<T>()).unwrap());
24.         }
25.     }
26. }
27. fn f3() {
28.     // let (y, x);
29.     // let (x, y);
30.     let x; let y;
31.     x = Hello::new("x", 13);
32.     y = MyBox2::new(Hello::new("y", &x));
33. }
34. fn main() {
35.     // f1();
36.     // f2();
37.     f3();
38. }

```

```

13-31 00:00:00 MyBox2T MyBoxT
PhantomDataT_pd RustMyBox2T
T T MyBox2T
[may_dangle] TMyBox2T

```

```

1726MyBox2TDrop[may_dangle]

```

```

00000000: 27 33 00 00 00 f3 00 00 00 00 00 00 00 00 00 00  x y 00 00 00 00 00
00000010: 00 drop 00 00 00 00 28 00 00 29 00 00 00 00 00 00 00 00 drop 00 00 00 00
00000020: 00 00 00

```

[illegible]

```
·[may_dangle] unsafe impl Drop {
}
```

```
· PhantomData[T] [0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [25] [26] [27] [28] [29] [30] [31] [32] [33] [34] [35] [36] [37] [38] [39] [40] [41] [42] [43] [44] [45] [46] [47] [48] [49] [50] [51] [52] [53] [54] [55] [56] [57] [58] [59] [60] [61] [62] [63] [64] [65] [66] [67] [68] [69] [70] [71] [72] [73] [74] [75] [76] [77] [78] [79] [80] [81] [82] [83] [84] [85] [86] [87] [88] [89] [90] [91] [92] [93] [94] [95] [96] [97] [98] [99] [100] [101] [102] [103] [104] [105] [106] [107] [108] [109] [110] [111] [112] [113] [114] [115] [116] [117] [118] [119] [120] [121] [122] [123] [124] [125] [126] [127] [128] [129] [130] [131] [132] [133] [134] [135] [136] [137] [138] [139] [140] [141] [142] [143] [144] [145] [146] [147] [148] [149] [150] [151] [152] [153] [154] [155] [156] [157] [158] [159] [160] [161] [162] [163] [164] [165] [166] [167] [168] [169] [170] [171] [172] [173] [174] [175] [176] [177] [178] [179] [180] [181] [182] [183] [184] [185] [186] [187] [188] [189] [190] [191] [192] [193] [194] [195] [196] [197] [198] [199] [200] [201] [202] [203] [204] [205] [206] [207] [208] [209] [210] [211] [212] [213] [214] [215] [216] [217] [218] [219] [220] [221] [222] [223] [224] [225] [226] [227] [228] [229] [230] [231] [232] [233] [234] [235] [236] [237] [238] [239] [240] [241] [242] [243] [244] [245] [246] [247] [248] [249] [250] [251] [252] [253] [254] [255] [256] [257] [258] [259] [260] [261] [262] [263] [264] [265] [266] [267] [268] [269] [270] [271] [272] [273] [274] [275] [276] [277] [278] [279] [280] [281] [282] [283] [284] [285] [286] [287] [288] [289] [290] [291] [292] [293] [294] [295] [296] [297] [298] [299] [300] [301] [302] [303] [304] [305] [306] [307] [308] [309] [310] [311] [312] [313] [314] [315] [316] [317] [318] [319] [320] [321] [322] [323] [324] [325] [326] [327] [328] [329] [330] [331] [332] [333] [334] [335] [336] [337] [338] [339] [340] [341] [342] [343] [344] [345] [346] [347] [348] [349] [350] [351] [352] [353] [354] [355] [356] [357] [358] [359] [360] [361] [362] [363] [364] [365] [366] [367] [368] [369] [370] [371] [372] [373] [374] [375] [376] [377] [378] [379] [380] [381] [382] [383] [384] [385] [386] [387] [388] [389] [390] [391] [392] [393] [394] [395] [396] [397] [398] [399] [400] [401] [402] [403] [404] [405] [406] [407] [408] [409] [410] [411] [412] [413] [414] [415] [416] [417] [418] [419] [420] [421] [422] [423] [424] [425] [426] [427] [428] [429] [430] [431] [432] [433] [434] [435] [436] [437] [438] [439] [440] [441] [442] [443] [444] [445] [446] [447] [448] [449] [450] [451] [452] [453] [454] [455] [456] [457] [458] [459] [460] [461] [462] [463] [464] [465] [466] [467] [468] [469] [470] [471] [472] [473] [474] [475] [476] [477] [478] [479] [480] [481] [482] [483] [484] [485] [486] [487] [488] [489] [490] [491] [492] [493] [494] [495] [496] [497] [498] [499] [500] [501] [502] [503] [504] [505] [506] [507] [508] [509] [510] [511] [512] [513] [514] [515] [516] [517] [518] [519] [520] [521] [522] [523] [524] [525] [526] [527] [528] [529] [530] [531] [532] [533] [534] [535] [536] [537] [538] [539] [540] [541] [542] [543] [544] [545] [546] [547] [548] [549] [550] [551] [552] [553] [554] [555] [556] [557] [558] [559] [560] [561] [562] [563] [564] [565] [566] [567] [568] [569] [570] [571] [572] [573] [574] [575] [576] [577] [578] [579] [580] [581] [582] [583] [584] [585] [586] [587] [588] [589] [590] [591] [592] [593] [594] [595] [596] [597] [598] [599] [600] [601] [602] [603] [604] [605] [606] [607] [608] [609] [610] [611] [612] [613] [614] [615] [616] [617] [618] [619] [620] [621] [622] [623] [624] [625] [626] [627] [628] [629] [630] [631] [632] [633] [634] [635] [636] [637] [638] [639] [640] [641] [642] [643] [644] [645] [646] [647] [648] [649] [650] [651] [652] [653] [654] [655] [656] [657] [658] [659] [660] [661] [662] [663] [664] [665] [666] [667] [668] [669] [670] [671] [672] [673] [674] [675] [676] [677] [678] [679] [680] [681] [682] [683] [684] [685] [686] [687] [688] [689] [690] [691] [692] [693] [694] [695] [696] [697] [698] [699] [700] [701] [702] [703] [704] [705] [706] [707] [708] [709] [710] [711] [712] [713] [714] [715] [716] [717] [718] [719] [720] [721] [722] [723] [724] [725] [726] [727] [728] [729] [730] [731] [732] [733] [734] [735] [736] [737] [738] [739] [740] [741] [742] [743] [744] [745] [746] [747] [748] [749] [750] [751] [752] [753] [754] [755] [756] [757] [758] [759] [760] [761] [762] [763] [764] [765] [766] [767] [768] [769] [770] [771] [772] [773] [774] [775] [776] [777] [778] [779] [780] [781] [782] [783] [784] [785] [786] [787] [788] [789] [790] [791] [792] [793] [794] [795] [796] [797] [798] [799] [800] [801] [802] [803] [804] [805] [806] [807] [808] [809] [810] [811] [812] [813] [814] [815] [816] [817] [818] [819] [820] [821] [822] [823] [824] [825] [826] [827] [828] [829] [830] [831] [832] [833] [834] [835] [836] [837] [838] [839
```

--	--	--	--	--	--	--	--	--

```

Rust13-32VecTLinkedListT

```

13-32 Vec T LinkedList T

```

1. pub struct Vec<T> {
2.     buf: RawVec<T>,
3.     len: usize,
4. }
5. pub struct RawVec<T, A: Alloc = Global> {
6.     ptr: Unique<T>,
7.     cap: usize,
8.     a: A,
9. }
10. pub struct Unique<T: ?Sized> {
11.     pointer: NonZero<*const T>,
12.     _marker: PhantomData<T>,
13. }
14. unsafe impl<#[may_dangle] T> Drop for Vec<T> {
15.     fn drop(&mut self) {
16.         unsafe {
17.             ptr::drop_in_place(&mut self[..]);
18.         }
19.     }
20. }
21. pub struct LinkedList<T> {
22.     head: Option<NonNull<Node<T>>>,
23.     tail: Option<NonNull<Node<T>>>,
24.     len: usize,
25.     marker: PhantomData<Box<Node<T>>>,
26. }
27. unsafe impl<#[may_dangle] T> Drop for LinkedList<T> {
28.     fn drop(&mut self) {
29.         while let Some(_) = self.pop_front_node() {}
30.     }
31. }

```

13-32 Vec<T> RawVec<T> T RawVec<T>
 Unique<T> Unique<T> **PhantomData<T>**
 drop

`Vec<T>` 的 `[may_dangle]` 属性表示该类型可能包含悬空指针

`LinkedList<T>` 和 `PhantomData<T>` 的 `[may_dangle]` 属性表示该类型可能包含悬空指针

`std::mem::forget` 函数

Rust 的 `std::mem::forget` 函数用于告诉编译器，该变量不会被使用，因此不需要进行垃圾回收。该函数接受一个 `C` 类型的变量，并将其从垃圾回收列表中移除。该函数在 `std::mem` 模块中定义。

13-33

13-33 使用 `std::mem::forget`

```
1. struct A;
2. struct B;
3. struct Foo {
4.     a: A,
5.     b: B
6. }
7. impl Foo {
8.     fn take(self) -> (A, B) {
9.         (self.a, self.b)
10.    }
11. }
12. fn main() {}
```

13-33 代码示例：使用 `std::mem::forget` 函数。该函数接受一个 `A` 类型的变量，并将其从垃圾回收列表中移除。该函数在 `std::mem` 模块中定义。

该函数在 `std::mem` 模块中定义。该函数接受一个 `A` 类型的变量，并将其从垃圾回收列表中移除。该函数在 `std::mem` 模块中定义。

13-34 使用 `std::mem::forget`

```

1. // 其余代码同上
2. impl Drop for Foo {
3.     fn drop(&mut self) {
4.         // 做一些事
5.     }
6. }

```

13-34 Foo Drop 13-35

13-35 13-34

```

error[E0509]: cannot move out of type `Foo`, which implements the `Drop` trait
--> src/main.rs:

```

```

| (self.a, self.b)
|     ^^^^^^ cannot move out of here

```

```

error[E0509]: cannot move out of type `Foo`, which implements the `Drop` trait
--> src/main.rs:

```

```

| (self.a, self.b)
|     ^^^^^^ cannot move out of here

```

13-35 Rust Foo Foo Drop Foo

Foo std mem forget 13-36

13-36 Foo take


```

1. use std::mem;
2. // 其余代码同上
3. impl Foo {
4.     fn take(mut self) -> (A, B) {
5.         let a = mem::replace(
6.             &mut self.a, unsafe { mem::uninitialized() }
7.         );
8.         let b = mem::replace(
9.             &mut self.b, unsafe { mem::uninitialized() }
10.        );
11.        mem::forget(self);
12.        (a, b)
13.    }
14. }

```

13-36 `Foo` `take` `mem::uninitialized` `mem::forget`

`mem::uninitialized` `unsafe` `take` `a` `b` “ ” **Rust** `a` `b` `FFI` `C`

`mem::forget` `Foo` “ ” `Foo` `forget` `unsafe` **Rust** `drop`

`std::mem::ManuallyDrop`

13-37

13-37 **ManuallyDrop**

```

1. use std::mem::ManuallyDrop;
2. struct Peach;
3. struct Banana;
4. struct Melon;
5. struct FruitBox {
6.     peach: ManuallyDrop<Peach>,
7.     melon: Melon,
8.     banana: ManuallyDrop<Banana>,
9. }
10. impl Drop for FruitBox {
11.     fn drop(&mut self) {
12.         unsafe {
13.             ManuallyDrop::drop(&mut self.peach);
14.             ManuallyDrop::drop(&mut self.banana);
15.         }
16.     }
17. }
18. fn main(){}

```

13-37 FruitBox peach banana ManuallyDropT ManuallyDrop drop peach banana

ManuallyDrop Rust 13-38 ManuallyDrop

13-38 ManuallyDropT

```

1.  #[allow(unions_with_drop_fields)]
2.  #[derive(Copy)]
3.  pub union ManuallyDrop<T>{ value: T }
4.  impl<T> ManuallyDrop<T> {
5.      pub const fn new(value: T) -> ManuallyDrop<T> {
6.          ManuallyDrop { value: value }
7.      }
8.      pub unsafe fn drop(slot: &mut ManuallyDrop<T>) {
9.          ptr::drop_in_place(&mut slot.value)
10.     }
11. }

```

13-38 `ManuallyDrop<T>` 实现 `ManuallyDrop<T>` 实现 Rust 的 `Drop` trait

实现 `ManuallyDrop` 的 `new` 和 `ManuallyDrop<T>` 的 `drop` 方法，使用 `std::mem::forget` 来忘记 `ManuallyDrop` 的 `new` 方法。
13-39

13-39 `forget` 方法

```

1.  pub fn forget<T>(t: T) {
2.      ManuallyDrop::new(t);
3.  }

```

13-39 `std::mem::forget` 方法

13.2.5 `NonNull<T>`

`NonNull<T>` 是一个不可变的 `*mut T` 指针，它是 `covariant` 且 `non-zero` 的。

`NonNull<T>` 在 `Unsafe Rust` 中，它支持 `*const T`、`*mut T`、`*const T`、`*mut T` 和 `*const T` 与 `&mut T`。

`NonNull` 在 `Unsafe` 中，它支持 `*const T`、`PhantomData<T>`、`*const T`、`*mut T`。

`NonNull<T>` 可以表示非空类型，`*mut T` 表示可变引用，`PhantomData<T>` 表示 phantom data，`drop` 表示析构。

`NonNull<T>`

13-40 `NonNull<T>`

13-40 `NonNull<T>` `NonZero`

```
1. pub struct NonNull<T: ?Sized> {
2.     pointer: NonZero<*const T>,
3. }
4. #[lang = "non_zero"]
5. #[derive(Copy, Clone, Eq, PartialEq, Ord, PartialOrd, Debug, Hash)]
6. pub struct NonZero<T: Zeroable>(pub(crate) T);
```

13-40 `NonNull<T>` `NonZero<*const T>` `*const T` `NonZero<*const T>` `NonNull<T>` `NonNull<T>` 13-32 `Unique<T>` `PhantomData<T>` `NonNull<T>` `T`

4-6 `NonZero<T>` Rust `core` `Zeroable` `T` `Null` `NonZero<T>` `[lang = "non_zero"]`

`NonNull<T>` 13-41

13-41 `NonNull<T>`

```

1. use std::ptr::{null, NonNull};
2. fn main(){
3.     let ptr : NonNull<i32> = NonNull::dangling();
4.     println!("{:p}", ptr); // 0x4
5.     let mut v = 42;
6.     let ptr : Option<NonNull<i32>> = NonNull::new(&mut v);
7.     println!("{:?}", ptr); // Some(0x7fff73406a78)
8.     println!("{:?}", ptr.unwrap().as_ptr()); // 0x7fff73406a78
9.     println!("{}", unsafe{ptr.unwrap().as_mut()}); // 42
10.    let mut v = 42;
11.    let ptr = NonNull::from(&mut v);
12.    println!("{:?}", ptr); // 0x7fff73406a7c
13.    let null_p: *const i32 = null();
14.    let ptr = NonNull::new(null_p as *mut i32);
15.    println!("{:?}", ptr); // None
16. }

```

13-41 3 4 **NonNull** **dangling** `Vec::new`
 NonNull `new`

5 7 **NonNull** **new** `Option`
 NonNull `T`

8 9 `as_ptr` `as_mut` `NonNull` `T`
 *mut T &mut T `as_mut`

10 12 **NonNull** **from** `NonNull`
 T

13 15 **null** `NonNull` `new`
 None

NonNull 13-42

13-42

```

1. use std::mem;
2. use std::ptr::NonNull;
3. struct Foo {
4.     a: *mut u64,
5.     b: *mut u64,
6. }
7. struct FooUsingNonNull {
8.     a: *mut u64,
9.     b: NonNull<*mut u64>,
10. }
11. fn main() {
12.     println!("*mut u64: {} bytes", mem::size_of::<*mut u64>());
13.     println!("NonNull<*mut u64>: {} bytes",
14.         mem::size_of::<NonNull<*mut u64>>());
15.     println!("Option<*mut u64>: {} bytes",
16.         mem::size_of::<Option<*mut u64>>());
17.     println!("Option<NonNull<*mut u64>>: {} bytes",
18.         mem::size_of::<Option<NonNull<*mut u64>>>());
19.     println!("Option<Foo>: {} bytes",
20.         mem::size_of::<Option<Foo>>());
21.     println!("Option<FooUsingNonNull>: {} bytes",
22.         mem::size_of::<Option<FooUsingNonNull>>());
23. }

```

13-42 Foo FooUsingNonNull NonNull *mut u64

main mem size_of 13-43

13-43

```

    13-43  *mut u64  NonNull*mut u64
    NonNull*mut u64  Option  NonNull*mut u64
    Option*mut u64  *mut u64

```

□□□□□□□□□□□□□□□□ FFI □ C □□ “□□□” □□□□□□□□

Unsafe Rust Rust 13-44

```
1. impl<T: Clone> Vec<T> {
2.     fn push_all(&mut self, to_push: &[T]) {
3.         self.reserve(to_push.len());
4.         unsafe {
5.             self.set_len(self.len() + to_push.len());
6.             for (i, x) in to_push.iter().enumerate() {
7.                 self.ptr().offset(i as isize).write(x.clone());
8.             }
9.         }
10.    }
11. }
```

13-44 Vec::T::push_all 3 reserve
unsafe write
unsafe

clone
clone push_all
clone
reserve set_len
Rust
C++ Rust

Rust **catch_unwind**
13-44 push_all clone
Rust
push_all catch_unwind

13.2.7

Unsafe Rust Rust std
alloc API

Rust Rust **1.28 jemalloc**
jemalloc Rust **1.28 jemalloc**
System

std alloc **GlobalAlloc** trait 13-45
13-45 GlobalAlloc trait

```
1. pub unsafe trait GlobalAlloc {  
2.     unsafe fn alloc(&self, layout: Layout) -> *mut u8;  
3.     unsafe fn dealloc(&self, ptr: *mut u8, layout: Layout);  
4.     // 其他方法省略  
5. }
```

13-45 GlobalAlloc **alloc** **dealloc**

GlobalAlloc unsafe trait

· 内存分配器

· `Layout` 内存布局

内存分配器 trait 13-46

13-46

```
1. use std::alloc::{GlobalAlloc, System, Layout};
2. struct MyAllocator;
3. unsafe impl GlobalAlloc for MyAllocator {
4.     unsafe fn alloc(&self, layout: Layout) -> *mut u8 {
5.         System.alloc(layout)
6.     }
7.     unsafe fn dealloc(&self, ptr: *mut u8, layout: Layout) {
8.         System.dealloc(ptr, layout)
9.     }
10. }
11. #[global_allocator]
12. static GLOBAL: MyAllocator = MyAllocator;
13. fn main() {
14.     // 此处 Vec 的内存会由 GLOBAL 全局分配器来分配
15.     let mut v = Vec::new();
16.     v.push(1);
17. }
```

13-46 `MyAllocator` 实现 `GlobalAlloc` trait
2-10 行 `System.alloc` 和 `System.dealloc` 调用
内存分配器

11-12 行 `[global_allocator]` 和 `GLOBAL`
`MyAllocator` 实现 `main` 函数 `Vec::T`
使用 `MyAllocator`

`[global_allocator]` 使用 `jemalloc`
13-47

13-47 `jemalloc`

```

1. extern crate jemallocator;
2. use jemallocator::Jemalloc;
3. #[global_allocator]
4. static GLOBAL: Jemalloc = Jemalloc;
5. fn main() {}

```

13-47 jemalloc Rust 1.28 jemalloc jemallocator extern crate jemallocator

[global_allocator] GLOBAL Jemalloc

Redox Rust **ralloc**

13.2.8

Unsafe Rust Safe Unsafe Rust [\[1\]](#)

- API
-
-

13.3

C Rust Rust

Common Lisp “Foreign Function Interface FFI” Haskell Python Ada “Language Bindings” Java FFI **JNI** Java Native Interface

FFI

13.3.1 FFI

FFI 是 Rust 中用于调用其他语言（如 C）的函数和库的接口。它允许 Rust 代码与 C 代码进行交互，从而可以利用 C 语言编写的库和函数。

在 Rust 中，FFI 主要用于调用 C 库。通过 FFI，Rust 代码可以调用 C 函数，并返回 C 函数的返回值。这通常用于调用操作系统 API、硬件驱动程序或其他底层库。

FFI 的基本用法如下：

首先，需要声明 C 函数的原型。这通常使用 `extern "C"` 关键字和 `abi = "C"` 属性来完成。

例如，假设我们有一个 C 函数 `add`，它接受两个整数并返回它们的和：

- 在 Rust 中，我们可以使用 `extern "C"` 和 `abi = "C"` 来声明这个函数。

然后，我们可以使用 `add` 函数来计算两个整数的和：

- 在 Rust 中，我们可以使用 `add` 函数来计算两个整数的和。

- 在 Rust 中，我们可以使用 `add` 函数来计算两个整数的和。

- 在 Rust 中，我们可以使用 `add` 函数来计算两个整数的和。

ABI 是 Application Binary Interface 的缩写，它定义了不同语言或库之间的接口。在 Rust 中，ABI 用于指定 Rust 代码与 C 代码之间的接口。通过 ABI，Rust 代码可以调用 C 函数，并返回 C 函数的返回值。

在 Rust 中，ABI 通常用于调用 C 库。通过 ABI，Rust 代码可以调用 C 函数，并返回 C 函数的返回值。这通常用于调用操作系统 API、硬件驱动程序或其他底层库。ABI 的定义通常位于 C 库的头文件中，Rust 代码通过包含这些头文件来使用 ABI。

Rust 提供了两种调用 C 库的方式：一种是使用 `extern "C"` 和 `abi = "C"` 来声明 C 函数，另一种是使用 `std::ffi::c` 模块来调用 C 函数。这两种方式都可以用于调用 C 库，但使用 `std::ffi::c` 模块的方式通常更简单，因为它不需要手动声明 C 函数的原型。

13-2 Rust FFI 调用 C 库的示例

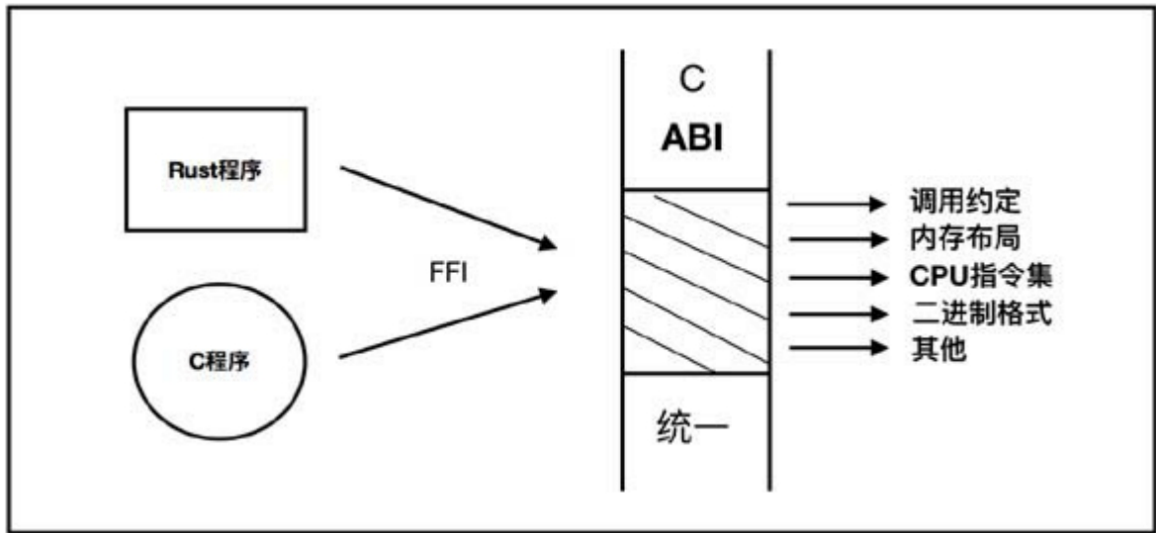


图13-2 Rust FFI与C的交互

Rust通过FFI与C交互，使用`extern`和`extern "C"`来指定FFI的接口。LLVM使用C-ABI。

Crate Type

ABI与Crate Type有关，Crate Type决定了ABI。

在Rust中，Crate Type决定了ABI。C/C++的Crate Type是`cdylib`或`staticlib`，而Rust的C/C++的Crate Type是`cdylib`。在Rust中，`extern crate`用于声明外部依赖，而`extern crate`用于声明外部依赖。

在Rust中，Crate Type决定了ABI。C/C++的Crate Type是`cdylib`或`staticlib`，而Rust的C/C++的Crate Type是`cdylib`。在Rust中，`extern crate`用于声明外部依赖，而`extern crate`用于声明外部依赖。

在Rust中，Crate Type决定了ABI。C/C++的Crate Type是`cdylib`或`staticlib`，而Rust的C/C++的Crate Type是`cdylib`。

在Rust中，Crate Type决定了ABI。C/C++的Crate Type是`cdylib`或`staticlib`，而Rust的C/C++的Crate Type是`cdylib`。

target triple **arm-unknown-linux-gnueabi**
target triple Triple {arch}-{vendor}-{sys}-{abi}
{arch}-{vendor}-{sys}-{abi}

arch arm vendor unknown sys Linux abi ABI gnueabi glibc C libc FPU arm-unknown-linux-gnueabi triple abi x86_64-apple-darwin wasm32-unknown-unknown [2]

Rust xargo std Rust xargo rustup

extern

Rust extern FFI

· extern extern Rust C

· extern Rust C extern C

Rust

extern Rust-ABI C-ABI extern ABI

· extern Rust ABI fn ABI

· extern C C-ABI “extern fn foo”

· extern system extern C Win32 stdcall

Rust extern ABI Reference [3] Rust ABI

· extern rust-intrinsic Rust ABI

· extern rust-call Fn call ABI

· extern platform-intrinsic ABI

extern

13.3.2 C/C++

C “” Rust

Rust C/C++ Rust Rust C/C++ Rust Rust C/C++ Rust C/C++ Rust

C-ABI Rust Ruby Python Node.js Rust FFI

Rust C

13-51 Rust C

13-51 Rust C

```
1. extern "C" {
2.     fn isalnum(input: i32) -> i32;
3. }
4. fn main() {
5.     unsafe {
6.         println!("Is 3 a number ? the answer is : {}", isalnum(3));
7.         // println!("Is 'a' a number ? ", isalnum('a'));
8.     }
9. }
```

13-51 1 extern C isalnum main C isalnum extern ABI C extern C-ABI

7 isalnum a extern Rust

Rust C++

Rust C++ C++ C-ABI

cargo bin rustcallcpp 13-52

13-52 rustcallcapp


```
$ cargo new --bin rustcallcpp
```

13-53 Cargo.toml

13-53 Cargo.toml

```
[package]
name = "rustcallcpp"
version = "0.1.0"
authors = ["blackanger <blackanger.z@gmail.com>"]
build = "build.rs"
edition = "2018"
[build-dependencies]
cc = "1.0"
```

13-53 build.rs build cc [\[4\]](#) Rust C/C++ C/C++ gcc g++ ar Rust crate build.rs Rust C/C++ cc gcc C/C++

rustcallcpp cpp_src C++ sorting.cpp sorting.h 13-54

13-54 rustcallcpp

```
├─ Cargo.lock
├─ Cargo.toml
├─ build.rs
├─ cpp_src
│   └─ sorting.cpp
│       └─ sorting.h
└─ src
    └─ main.rs
```

sorting.cpp Rust C++ 13-55

13-55 sorting.cpp

```

1. #include "sorting.h"
2. void interop_sort(int numbers[], size_t size)
3. {
4.     int* start = &numbers[0];
5.     int* end = &numbers[0] + size;
6.     std::sort(start, end, [](int x, int y) { return x > y; });
7. }

```

sorting.cpp implements interop_sort using the C++ sort function.

sorting.h defines the C function 13-56

13-56 sorting.h

```

1. #ifndef __SORTING_H__
2. #define __SORTING_H__ "sorting.h"
3. #include <iostream>
4. #include <functional>
5. #include <algorithm>
6. #ifdef __cplusplus
7. extern "C" {
8. #endif
9. void interop_sort(int[], size_t);
10. #ifdef __cplusplus
11. }
12. #endif
13. #endif

```

sorting.h defines extern C interop_sort C function Rust

src/main.rs implements 13-57

13-57 src/main.rs

```

1. #[link(name = "sorting", kind = "static")]
2. extern {

```

```

3.     fn interop_sort(arr: &[i32;10], n: u32);
4. }
5. pub fn sort_from_cpp(arr: &[i32;10], n: u32) {
6.     unsafe {
7.         interop_sort(arr, n);
8.     }
9. }
10. fn main() {
11.     let my_arr: [i32; 10] = [10, 42, -9, 12, 8, 25, 7, 13, 55, -1];
12.     println!("Before sorting...");
13.     println!("{:?}", my_arr);
14.     sort_from_cpp(&my_arr, 10);
15.     println!("After sorting...");
16.     println!("{:?}", my_arr);
17. }

```

13-57 1 **link** **name=** **sorting** **kind=** **static** **libsorting** [\[5\]](#) Rust

2 4 **extern** **interop_sort** C++

5 9 Rust **sort_from_cpp** C++ **interop_sort** **main**

C++ Rust **build.rs** 13-58

13-58 build.rs

```

1. extern crate cc;
2. fn main() {
3.     cc::Build::new()
4.         .cpp(true)
5.         .warnings(true)
6.         .flag("-Wall")
7.         .flag("-std=c++14")
8.         .flag("-c")
9.         .file("cpp_src/sorting.cpp")
10.        .compile("sorting");
11. }

```

13-58 cc crate cc crate cpp_src C++

· **g++-Wall-std=c++14-c sorting.cpp** g++ sorting.cpp

· **ar rc libsorting.a sorting.o** ar libsorting.a

cargo run 13-59

13-59

Before sorting...

[10, 42, -9, 12, 8, 25, 7, 13, 55, -1]

After sorting...

[-9, -1, 7, 8, 10, 12, 13, 25, 42, 55]

C++ main.rs 10 10 Rust C++ Rust target/debug/build cpp_src/sorting.o libsorting.a

cc build.rs Command new g++ C++ cc cc C

C Rust

C Rust cargo callrust 13-60

13-60 callrust

```
$ cargo new --lib callrust
```

13-61

13-61 callrust

```
├─ Cargo.toml
├─ c_src
│   └─ main.c
├─ makefile
└─ src
    ├── callrust.h
    └─ lib.rs
```

13-61

- c_src C
- c_src/main.c C
- src/callrust.h Rust C
- makefile

Cargo.toml 13-62

13-62 Cargo.toml

```
[dependencies]
libc="0.2"

[lib]
name = "callrust"
crate-type = ["staticlib", "cdylib"]
```

Cargo.toml **libc** libc Rust C Rust

Rust **callrust** **staticlib** **cdylib** C-ABI

src/lib.rs 13-63

13-63 src/lib.rs

```
1. use libc;
2. #[no_mangle]
3. pub extern fn print_hello_from_rust() {
4.     println!("Hello from Rust");
5. }
```

13-63 lib.rs print_hello_from_rust pub
extern C-ABI

[no_mangle] Rust Rust C

src/callrust.h print_hello_from_rust C Rust 13-64

13-64 src/callrust.h

```
1. void print_hello_from_rust();
```

C c_src/main.c 13-65

13-65 c_src/main.c

```
1. #include "callrust.h"
2. #include <stdio.h>
3. #include <stdint.h>
4. #include <inttypes.h>
5. int main (void) {
6.     print_hello_from_rust();
7. }
```

13-65 callrust.h main print_hello_from_rust

makefile make 13-66

13-66 makefile

```

1.  GCC_BIN ?= $(shell which gcc)
2.  CARGO_BIN ?= $(shell which cargo)
3.  run: clean build
4.      ./c_src/main
5.  clean:
6.      $(CARGO_BIN) clean
7.      rm -f ./c_src/main
8.  build:
9.      $(CARGO_BIN) build
10.   $(GCC_BIN) -o ./c_src/main ./c_src/main.c -Isrc -L ./target/debug
    -lcallrust

```

13-66 make run clean build build

- cargo build Rust C-ABI

- gcc C Rust main

makefile tab

make make run

13-67

13-67

```

/usr/bin/gcc -o ./c_src/main ./c_src/main.c -Isrc -L ./target/debug
-lcallrust

```

```

./c_src/main

```

```

Hello from Rust

```

13-67 make print_hello_from_rust

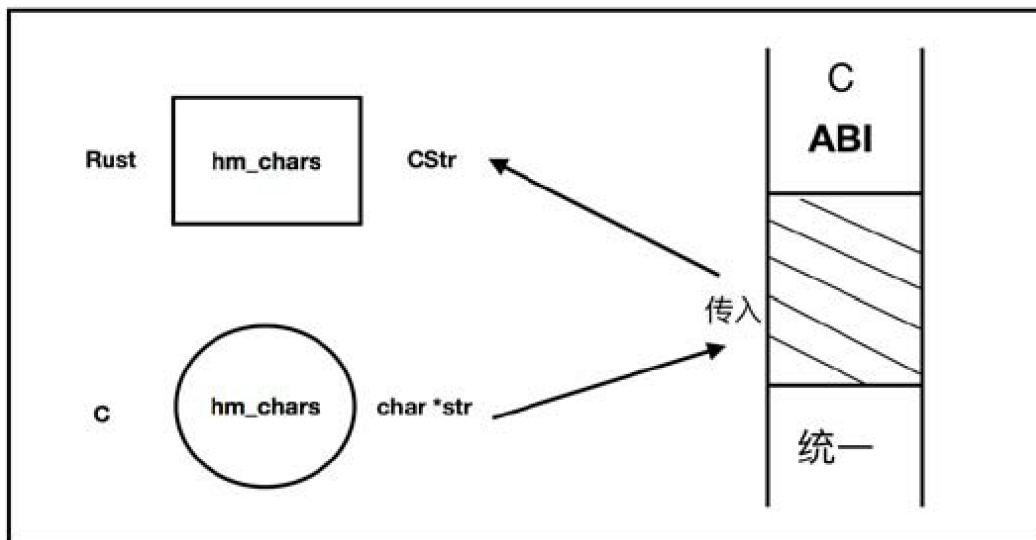
Rust C Rust C

callrust Rust C callrust src/lib.rs 13-68

13-68 src/lib.rs

```
1. use libc::{c_char, c_uint};
2. use std::ffi::CStr;
3. #[no_mangle]
4. pub extern fn hm_chars(s: *const c_char) -> c_uint {
5.     let c_str = unsafe {
6.         assert!(!s.is_null());
7.         CStr::from_ptr(s)
8.     };
9.     let r_str = c_str.to_str().unwrap();
10.    r_str.chars().count() as c_uint
11. }
```

13-68 Rust C "n" char*str Rust 13-4



13-4 C to Rust

```
C_hm_chars char*str Rust
C char*str
```

```
Rust Char C Char Rust Char Unicode
C Char Rust std os raw C
c_char i8
hm chars std os raw c char
```


callrust 使用 libc 的 C 函数。使用 libc 的 c_char 类型，std::os::raw 使用 libc 的 raw 类型。使用 libc 的 std 函数。使用 libc 的 std 函数。

Rust 使用 Rust 的 Rust 的 Rust 的 std::ffi::CStr 类型，使用 "\n" 换行符。使用 5 和 8 的 CStr 从 c_char 转换为 Rust 的 CStr 类型。

使用 9 的 CStr 类型，使用 &str1 的 10 个 chars 类型。使用 Rust 的 count 函数。使用 C 的 C-ABI 的 libc 的 c_uint 类型。

使用 callrust.h 的 hm_chars 函数。使用 C 的 13-69 函数。

13-69 使用 src/callrust.h 函数。

1. #include <inttypes.h>
2. uint32_t hm_chars(const char *str);

13-69 使用 hm_chars 函数。使用 C 的 uint32_t 类型。使用 inttypes.h 函数。

c_str/main.c 使用 main 函数。使用 hm_chars 函数。使用 13-70 函数。

13-70 使用 c_str/main.c 的 main 函数。

1. uint32_t count = hm_chars("The taö of Rust");
2. printf("%d\n", count);

13-70 使用 main 函数。使用 1 的 hm_chars 函数。使用 count 函数。使用 2 的 count 函数。

使用 callrust 函数。使用 make 函数。使用 13-71 函数。

13-71 使用 make 函数。

//此处省略掉之前函数的输出

15

□□□□□□□□□□□□

□□□□src/lib.rs□□□□□□□□□□□□□□□□13-72□□□

□□□□13-72□□src/lib.rs□□□□□□□□

```
1. use std::ffi::{CStr, CString};
2. use std::iter;
3. #[no_mangle]
4. pub extern fn batman_song(length: c_uint) -> *mut c_char {
5.     let mut song = String::from("boom ");
6.     song.extend(iter::repeat("nana ").take(length as usize));
7.     song.push_str("Batman! boom");
8.     let c_str_song = CString::new(song).unwrap();
9.     c_str_song.into_raw()
10. }
```

□□□□13-72□□□□□□□□□□batman_song□□□□□□□□□□□□“boom
nana nana nana Batman boom”□□□□□□“□□□□□ ”□□□□□□□□
“nana”□□□□□□□□□□batman_song□□□□□□□□□□

□□□□C□□□□□□□□□□C□□□□□□□□□□□□□□□□Rust□□□String□□□□□□
String□□□□□□□□□□□□□□□□std□□ffi□□□□**CString** □□□String□□□C-
ABI□□□□□□□□□□□□**CStr** □□□□□□□□String□□□□□□□□□□□□□□□□□
CString□□□□□8□□□9□□□□□□□String□□CString□□□□□□□□□□
into_raw□□□□□C□□□□□□□

□□CString□□□
□□□□□□□□□□Rust□□C□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□C□□□□□□□□□□□□13-73□□□

□□□□13-73□□src/lib.rs□□□□□□□□□□free_song

```

1.  #[no_mangle]
2.  pub extern fn free_song(s: *mut c_char) {
3.      unsafe {
4.          if s.is_null() { return }
5.          CString::from_raw(s)
6.      };
7.  }

```

13-73 调用 `free_song` 函数，参数为 `*mut c_char`，表示一个 C 字符串的指针。该函数在 `CString::from_raw` 方法中定义，用于释放 Rust 字符串占用的内存。

13-74 在 `src/callrust.h` 文件中定义 C 函数原型。

13-74 在 `src/callrust.h` 文件中定义 C 函数原型。

```

1.  //省略其他函数声明
2.  char * batman_song(uint8_t length);
3.  void free_song(char *);

```

13-75 在 `c_src/main.c` 文件中定义 `main` 函数。

13-75 在 `c_src/main.c` 文件中定义 `main` 函数。

```

1.  // 省略其他代码
2.  char *song = batman_song(5);
3.  printf("%s\n", song);
4.  free_song(song);

```

13-75 在 `batman_song` 函数中调用 `free_song` 函数，用于释放分配的内存。该函数在 `RustString` 结构中定义，用于释放 Rust 字符串占用的内存。

13-76 在 `make` 文件中定义 `main` 函数。

13-76 在 `make` 文件中定义 `main` 函数。

//此处省略之前函数的输出

boom nana nana nana nana nana Batman! boom

13-76 在 `make` 文件中定义 `main` 函数。

Rust C 13-77
src/lib.rs

```
1. use std::slice;
2. #[no_mangle]
3. pub extern fn sum_of_even(n: *const c_uint, len: c_uint) -> c_uint
4. {
5.     let numbers = unsafe {
6.         assert!(n.is_null());
7.         slice::from_raw_parts(n, len as usize)
8.     };
9.     let sum = numbers.iter()
10.        .filter(|&v| v % 2 == 0)
11.        .fold(0, |acc, v| acc + v);
12.     sum as c_uint
13. }
```

C Rust 13-77
sum_of_even *const c_uint c_uint
5 8 slice from_raw_parts C
9 11
src/callrust.h 13-78
13-78 src/callrust.h

```
1. //省略其他函数声明
2. #include <stdio.h>
3. uint32_t sum_of_even(const uint32_t *numbers, size_t length);
```

13-78 stdio.h size_t
c_src/main.c main 13-79
13-79 c_src/main.c main

```

1. // 省略其他代码
2. uint32_t numbers[6] = {1,2,3,4,5,6};
3. uint32_t sum = sum_of_even(numbers, 6);
4. printf("%d\n", sum);

```

编译make

C Rust 编译选项 C 编译选项

C Rust 编译选项 13-80

13-80 src/lib.rs

```

1. #[repr(C)]
2. pub struct Tuple {
3.     x: c_uint,
4.     y: c_uint,
5. }
6. impl From<(u32, u32)> for Tuple {
7.     fn from(tup: (u32, u32)) -> Tuple {
8.         Tuple { x: tup.0, y: tup.1 }
9.     }
10. }
11. impl From<Tuple> for (u32, u32) {
12.     fn from(tup: Tuple) -> (u32, u32) {
13.         (tup.x, tup.y)
14.     }
15. }
16. fn compute_tuple(tup: (u32, u32)) -> (u32, u32) {
17.     let (a, b) = tup;
18.     (b+1, a-1)
19. }
20. #[no_mangle]
21. pub extern fn flip_things_around(tup: Tuple) -> Tuple {
22.     compute_tuple(tup.into()).into()
23. }

```

13-80 1 5 Tuple

[repr(C)] C ABI C Rust


```

1. use std::collections::HashMap;
2. pub struct Database {
3.     data: HashMap<String, u32>,
4. }
5. impl Database {
6.     fn new() -> Database {
7.         Database {
8.             data: HashMap::new(),
9.         }
10.    }
11.    fn insert(&mut self) {
12.        for i in 0..100000 {
13.            let zip = format!("{:05}", i);
14.            self.data.insert(zip, i);
15.        }
16.    }
17.    fn get(&self, zip: &str) -> u32 {
18.        self.data.get(zip).cloned().unwrap_or(0)
19.    }
20. }

```

13-83 Database HashMap String u32
 new insert get new
 Database insert 100000
100086=100086 "get

Database C
[repr C] C-ABI C

C 13-84
13-84 **src/lib.rs**

```

1.  #[no_mangle]
2.  pub extern fn database_new() -> *mut Database {
3.      Box::into_raw(Box::new(Database::new()))
4.  }
5.  #[no_mangle]
6.  pub extern fn database_insert(ptr: *mut Database) {
7.      let database = unsafe {
8.          assert!(!ptr.is_null());
9.          &mut *ptr
10.     };
11.     database.insert();
12. }
13. #[no_mangle]
14. pub extern fn database_query(ptr: *const Database,
15.     zip: *const c_char) -> c_uint
16. {
17.     let database = unsafe {
18.         assert!(!ptr.is_null());
19.         &*ptr
20.     };
21.     let zip = unsafe {
22.         assert!(!zip.is_null());
23.         CStr::from_ptr(zip)
24.     };
25.     let zip_str = zip.to_str().unwrap();
26.     database.get(zip_str)
27. }
28. #[no_mangle]
29. pub extern fn database_free(ptr: *mut Database) {
30.     if ptr.is_null() { return }
31.     unsafe { Box::from_raw(ptr); }
32. }

```

□ □ □ □ □ 13-84 □ □ □ □ □ □ □ □ □ □ □ □ database_new □
 database_insert □ database_query □ □ □ □ □ Database □ □ □ □ □ new □

insert[]get[]

```
24 database_new() *mut Database Database
Database() C Database new() Box new()
Box into_raw() *mut Database Database
C
```

```
527 database_insert() database_query()
HashMap<String, u32> *mut Database Database
```

```
2832 database_free() Rust Rust
C Box Box Box Box
```

lib/callrust.h 13-85

13-85 src/callrust.h

1. //省略其他函数声明
2. typedef struct database_S database_t;
3. database_t * database_new(void);
4. void database_free(database_t *);
5. void database_insert(database_t *);
6. uint32_t database_query(const database_t *, const char *zip);

```
13-852 database_S database_t
```

c_src/main.c main 13-86

13-86 c_src/main.c main

1. // 省略其他代码
2. database_t *database = database_new();
3. database_insert(database);
4. uint32_t pop1 = database_query(database, "10186");
5. uint32_t pop2 = database_query(database, "10852");
6. database_free(database);
7. printf("%d\n", pop2 - pop1);


```

1.  require 'ffi'
2.  class Database < FFI::AutoPointer
3.    def self.release(ptr)
4.      Binding.free(ptr)
5.    end
6.    def insert
7.      Binding.insert(self)
8.    end
9.    def query(zip)
10.     Binding.query(self, zip)
11.  end
12.  module Binding
13.    extend FFI::Library
14.    ffi_lib "../target/debug/libcallrust.dylib"
15.    attach_function :new, :database_new, [], Database
16.    attach_function :free, :database_free, [Database], :void
17.    attach_function :insert, :database_insert, [Database], :void
18.    attach_function :query, :database_query,
19.      [Database, :string], :uint32
20.  end
21. end
22. database = Database::Binding.new
23. database.insert
24. pop1 = database.query("10186")
25. pop2 = database.query("10852")
26. puts pop2 - pop1

```

13-87 Ruby Rust Database ffi
gem

211 FFI AutoPointer Database FFI
AutoPointer self.release Ruby GC
 Rust Rust Database
self.release Rust insert query
 Rust

2012年12月 Binding 拡張 Mixin FFI Library libffi Rust 14 **ffi_lib** Rust [\[11\]](#) **attach_function** Rust Ruby

2012年12月26日 Ruby Ruby

Rust Ruby

・ **Ruru** **Rutie** Rust Ruby Ruby

・ **Helix** Ruby Ruby Ruby Ruby Helix `helix-rails gem` Rails Rails Helix Ruby

Rust Ruby Ruby Ruby GC Ruby Rust Ruby

Python

Python Rust Python 3 CTypes libffi CTypes C-ABI

`callrust` Python `database.py` 13-88

13-88 **Python/database.py** Python

```

1. #!/usr/bin/env python3
2. import sys, ctypes
3. from ctypes import c_char_p, c_uint32, Structure, POINTER
4. prefix = {'win32': ''}.get(sys.platform, '../target/debug/lib')
5. extension = {'darwin': '.dylib', 'win32': '.dll'} \
6.     .get(sys.platform, '.so')
7. class DatabaseS(Structure):
8.     pass
9. lib = ctypes.cdll.LoadLibrary(prefix + "callrust" + extension)
10. lib.database_new.restype = POINTER(DatabaseS)
11. lib.database_free.argtypes = (POINTER(DatabaseS), )
12. lib.database_insert.argtypes = (POINTER(DatabaseS), )
13. lib.database_query.argtypes = (POINTER(DatabaseS), c_char_p)
14. lib.database_query.restype = c_uint32
15. class Database:
16.     def __init__(self):
17.         self.obj = lib.database_new()
18.     def __enter__(self):
19.         return self
20.     def __exit__(self, exc_type, exc_value, traceback):
21.         lib.database_free(self.obj)
22.     def insert(self):
23.         lib.database_insert(self.obj)
24.     def query(self, zip):
25.         return lib.database_query(self.obj, zip.encode('utf-8'))
26. with Database() as database:
27.     database.insert()
28.     pop1 = database.query("10186")
29.     pop2 = database.query("10852")
30.     print(pop2 - pop1)

```

13-8723 CTypes

46 Rust

78 DatabaseS

914 CTypes POINTER DatabaseS DatabaseS POINTER

1525 Database insert query
__enter__ __exit__ Database with with
with __enter__
as __exit__

2630 with
__exit__ Rust database_free

Ruby Python Rust
· **rust-cpython** Python Rust Python 2.7
Python 3.3+

· **PyO3** Python Rust rust-cpython
rust-cpython

PyO3
Node.js
Node.js I/O
URL CPU Node.js C++
Rust

Node.js V8.0 **NAN** Native Abstractions for
Node.js API V8.0 **N-API**
NAN N-API C-API Node.js
JavaScript

callrust Node.js/database.js 13-89

13-89 **Node.js/database.js**

```

1.  const ffi = require('ffi-napi');
2.  const lib = ffi.Library('../target/debug/libcallrust.dylib', {
3.      database_new: ['pointer', []],
4.      database_free: ['void', ['pointer']],
5.      database_insert: ['void', ['pointer']],
6.      database_query: ['uint32', ['pointer', 'string']],
7.  });
8.  const Database = function() {
9.      this.ptr = lib.database_new();
10. };
11. Database.prototype.free = function() {
12.     lib.database_free(this.ptr);
13. };
14. Database.prototype.insert = function() {
15.     lib.database_insert(this.ptr);
16. };
17. Database.prototype.query = function(zip) {
18.     return lib.database_query(this.ptr, zip);
19. };
20. const database = new Database();
21. try {
22.     database.insert();
23.     const pop1 = database.query("10186")
24.     const pop2 = database.query("10852")
25.     console.log(pop2 - pop1);
26. }finally {
27.     database.free();
28. }

```

13-8917 **ffi-napi** Rust
 Node.js ffi-napi N-API [\[12\]](#)
 8 10 JavaScript Database
 lib.database_new ptr

11 19 prototype Database free insert query Rust database_free database_insert database_query

20 28 Database try finally database.free Rust

Node.js Rust Neon

Neon Rust Node.js JavaScript Neon NAN N-API

Ruby Python Node.js Erlang BEAM Elixir Rust

Elixir Erlang NIF Native Implemented Function NIF Erlang C libffi C NIF C/C++ Erlang Erlang Erlang Erlang Rust NIF

Rust NIF Rustler BEAM Erlang Elixir Elixir Rustler

FFI Rust Java Swfit Lua Haskell OCmal

- **jni-rs** Rust JNI Java
 - **rlua** Rust Lua Lua
 - **rmal** Rust OCmal OCmal
-

13.4 Rust→WebAssembly

WebAssembly





WebAssembly Google Microsoft Mozilla
Web C/C++/Rust WebAssembly IR
WebAssembly Assembly
WebAssembly Web

WebAssembly JavaScript API
JavaScript WebAssembly
WebAssembly/

WebAssembly↔JavaScript↔

- WebAssembly `WebAssembly` `JavaScript`
- WebAssembly `JavaScript` CPU 64
- `WebAssembly` `Rust` `wasm-gc` `wasm`
- WebAssembly `JavaScript` `JavaScript` `JavaScript` `GC`

WebAssembly
DOM
Reference Types [13]
Host Bindings [14] WebAssembly
JavaScript+DOM

WebAssembly  **Web**  **Web**  **Web**  **Web** **Web**


```

├── README.md
├── build.ts
├── package.json
├── src
│   ├── main.html
│   ├── main.js
│   └── main.wat

```

13-90 build.ts wasm out src
 main.wat WebAssembly
 main.wasm main.js wasm main.html

main.wat 13-91
 13-91 main.wat

```

1. (module
2.   (func $add (param $lhs i32) (param $rhs i32) (result i32)
3.     get_local $lhs
4.     get_local $rhs
5.     i32.add)
6.   (export "add" (func $add))
7. )

```

13-91 S
 13-91
 module func export module func
 export

2 func \$add \$
 / func
 param i32 result
 i32

3 5 get_local i32.add
 i32 WebAssembly [16]
 wasm get_local
 i32.add i32

export export "add" JavaScript

WebAssemblymain.js13-92

13-92main.js

```
1. fetch('../out/main.wasm').then(response =>
2.   response.arrayBuffer()
3. ).then(bytes => WebAssembly.instantiate(bytes))
4.   .then(results => {
5.     instance = results.instance;
6.     document.getElementById("container").innerText =
7.       instance.exports.add(1,1);
8.   }).catch(console.error);
```

13-92fetch out/main.wasm
ArrayBuffer XMLHttpRequestwasm

WebAssembly.instantiate
addJavaScriptinstance.exports.add

webassembly.studioIDEbuild&run
"2"

WebAssemblyJavaScript

WebAssemblyi32i64f32f64
WebAssemblyWebAssembly
JavaScript WebAssembly.Memory

main.wat"Hi WASM"13-93

13-93main.wat

```

1.  (module
2.      (import "console" "log" (func $log (param i32 i32)))
3.      (import "js" "mem" (memory 1))
4.      (data (i32.const 0) "Hi WASM,")
5.      (data (i32.const 8) "I'm Coming")
6.      (func (export "writeHi")
7.          i32.const 0
8.              i32.const 18
9.              call $log)
10. )

```

13-93 import data func
 2 3 import JavaScript
 WebAssembly2 console.log \$log 3
 JavaScript 1 64KB
 WebAssembly

4 data "i32 const 0"
 0 5 data
 8 "Hi WASM" 8 0
 8

6 9 writeHi JavaScript call
 JavaScript \$log

main.js 13-94

13-94 main.js

```

1. var memory = new WebAssembly.Memory({ initial : 1 });
2. function consoleLogString(offset, length) {
3.     var bytes = new Uint8Array(memory.buffer, offset, length);
4.     var string = new TextDecoder('utf8').decode(bytes);
5.     console.log(string);
6. }
7. var importObject = {
8.     console: {
9.         log: consoleLogString
10.    },
11.    js: {
12.        mem: memory
13.    }
14. };
15. WebAssembly.instantiateStreaming(
16.    fetch('../out/main.wasm'), importObject
17. ).then(obj => {
18.    obj.instance.exports.writeHi();
19. });

```

13-94 1 **WebAssembly.Memory** wasm 1

2 6 consoleLogString offset length **Uint8Array** **TextDecoder** wasm JavaScript

7 14 importObject JavaScript wasm log wat import

15 19 **WebAssembly.instantiateStreaming** wasm

webassembly.studio IDE build&run

“**Hi WASM I’m Coming**” [\[17\]](#)

このファイルは、wasm 形式のコードを生成するためのビルドスクリプトです。i32call_indirect を使用して、共有ライブラリを呼び出します。

このwasmファイルは、webassembly.studioというIDEで、src/main.wat、src/shared0.wat、src/shared1.watをビルドします。build.tsは13-95で定義されています。

13-95 build.ts

```
1. gulp.task("build", async () => {
2.   const data0 = await Service.assembleWat(
3.     project.getFile("src/shared0.wat").getData()
4.   );
5.   const outWasm0 = project.newFile(
6.     "out/shared0.wasm", "wasm", true
7.   );
8.   outWasm0.setData(data0);
9.   const data1 = await Service.assembleWat(
10.    project.getFile("src/shared1.wat").getData()
11.  );
12.  const outWasm1 = project.newFile("out/shared1.wasm", "wasm", true);
13.  outWasm1.setData(data1);
14. });
```

build.tsは、gulp.taskを使用して、13-95で定義された共有ライブラリ（shared0.wat、shared1.wat）をWebAssemblyに変換します。

shared0.watは13-96で定義されています。

13-96 shared0.wat

```

1. (module
2.   (import "js" "memory" (memory 1))
3.   (import "js" "table" (table 1 anyfunc))
4.   (elem (i32.const 0) $shared0func)
5.   (func $shared0func (result i32)
6.     i32.const 0
7.     i32.load)
8. )

```

13-96 2 3 JavaScript 1 anyfunc “ ”

4 elem \$shared0func 0 elem data

5 7 \$shared0func 0 i32.load 0

shared1.wat 13-97

13-97 shared1.wat

```

1. (module
2.   (import "js" "memory" (memory 1))
3.   (import "js" "table" (table 1 anyfunc))
4.   (type $void_to_i32 (func (result i32)))
5.   (func (export "doIt") (result i32)
6.     i32.const 0
7.     i32.const 42
8.     i32.store
9.     i32.const 0
10.    call_indirect (type $void_to_i32))
11. )

```

13-97 JavaScript 4 type \$void_to_i32

5 10 JavaScript doIt 6 8 “i32.store i32.const 0 i32.const 42” 42 0

· **C/C++** → **EmScripten** → **wasm** → **EmScripten** →
→ **LLVM** → **LLVM** → **asm.js** → **C/C++** →
→ **LLVM** → **Clang** → **LLVM IR** → **EmScripten** →
asm.js → **WebAssembly** → **Binaryen** → **asm.js** → **wasm** →
→ **asm.js** → **JavaScript** → **WebAssembly** →
→ **wasm** → **asm.js**

· **Rust** → **wasm**

➤ **wasm32-unknown-unknown** → **LLVM**
WebAssembly Backend → **lld**

➤ **wasm32-unknown-emscripten** → **EmScripten** →
C/C++

→ **wasm32-unknown-unknown** → **Rust** → **wasm** →
→ **wasm** → **rustup** → **13-99**

→ **13-99** → **rustup**

```
$ rustup toolchain install nightly
```

```
$ rustup target add wasm32-unknown-unknown --toolchain nightly
```

→ **13-99** → **rustup** → **Nightly** → **rustup**
target add → **wasm32-unknown-unknown** → **rustup** →

→ **cargo new--lib** → **hello_wasm** →
Cargo.toml → **lib** → **13-100**

→ **13-100** → **Cargo.toml**

```
[lib]
```

```
path = "src/lib.rs"
```

```
crate-type = ["cdylib"]
```

→ **src/lib.rs** → **13-101**

→ **13-101** → **src/lib.rs**

```

1.  #[link(wasm_import_module = "env")]
2.  extern "C" {
3.      pub fn logit();
4.      pub fn hello(ptr: *const u8, len: u32);
5.  }
6.  #[no_mangle]
7.  pub extern "C" fn add_one(x: i32) {
8.      unsafe {
9.          logit();
10.         let msg = format!("Hello world: {}", x + 1);
11.         hello(msg.as_ptr(), msg.len() as u32);
12.     }
13. }

```

13-101 15 extern C JavaScript
logit hello **logit** JavaScript console.log
hello Rust wasm
JavaScript 1 **[link wasm_import_module=**
env] extern wasm env
env

Rust C-ABI **LLVM WebAssembly Backend** lld wasm extern

6 12 **[no_mangle]** pub extern C
add_one i32
logit hello 9 msg msg.as_ptr
hello

hello_wasm hello.html hello.js
hello.html 13-102

13-102 **hello.html**

```

1. <!DOCTYPE html>
2. <html lang="en">
3.   <head>
4.     <meta charset="utf-8">
5.     <title>hello wasm</title>
6.     <script src="./hello.js"></script>
7.   </head>
8. </html>

```

13-102 HTML 6 hello.js

13-103

```

1. var mod;
2. var imports = {
3.   logit: () => {
4.     console.log('this was invoked by Rust, written in JS');
5.   },
6.   hello: (ptr, len) => {
7.     var buf = new Uint8Array(
8.       mod.instance.exports.memory.buffer, ptr, len
9.     )
10.    var msg = new TextDecoder('utf8').decode(buf);
11.    alert(msg);
12.  }
13. }
14. fetch('output/small_hello.wasm')
15.   .then(response => response.arrayBuffer())
16.   .then(bytes => WebAssembly.instantiate(bytes, {env: imports}))
17.   .then(module => {
18.     mod = module;
19.     module.instance.exports.add_one(41);
20.   });

```

13-103 1 mod wasm

2 13 imports logit hello

hello ptr Uint8Array TextDecoder ptr JavaScript Uint8Array **mod.instance.exports.memory.buffer** ArrayBuffer

1420 fetch wasm arrayBuffer WebAssembly.instantiate imports env wasm add_one

Rust wasm **hello_wasm** output wasm wasm 13-104

13-104 wasm

```
$ cargo build --target wasm32-unknown-unknown
$ cp target/wasm32-unknown-unknown/debug/hello_wasm.wasm output
$ wasm-gc output/hello_wasm.wasm output/small_hello.wasm
```

13-104 cargo build wasm32-unknown-unknown target target/wasm32-unknown-unknown/debug hello.wasm output **wasm-gc** output/hello_wasm.wasm output/small_hello.wasm make

cargo install wasm-gc wasm ld LTO wasm-gc

[\[19\]](#) hello_wasm/hello.html **“Hello world”** 42 Rust wasm

13.4.3 WebAssembly

Rust wasm WebAssembly Rust wasm-bindgen Rust wasm

· **wasm-bindgen** Javascript Rust wasm Rust Javascript wasm

· **wasm-pack** Rust wasm npm
npm node.js JavaScript wasm-pack JavaScript
npm

· **cargo-generate** wasm-bindgen wasm-pack

Rust Rust WebAssembly
Rust wasm

wasm-bindgen JavaScript Rust crate
wasm-bindgen **js-sys** JavaScript API
wasm_bindgen js **web-sys**
Web API

cargo install cargo-generate cargo-generate cargo-generate wasm-bindgen
13-105

13-105 cargo-generate

```
$ cargo-generate --git \  
https://github.com/ashleygwilliams/wasm-pack-template
```

Cargo.toml wasm-bindgen
wasm-bindgen Nightly wasm-bindgen-cli
13-106

13-106 wasm-bindgen-cli

```
$ cargo +nightly install wasm-bindgen-cli
```

wasm-bindgen-cli wasm-bindgen wasm
wasm-pack 13-107

13-107 wasm-pack wasm-pack

```
$ cargo install wasm-pack
```

```
$ wasm-pack build
```

13-107 wasm-pack
wasm-pack build JavaScript
wasm npm

Rust WebAssembly Rust wasm-bindgen wasm-pack WebAssembly

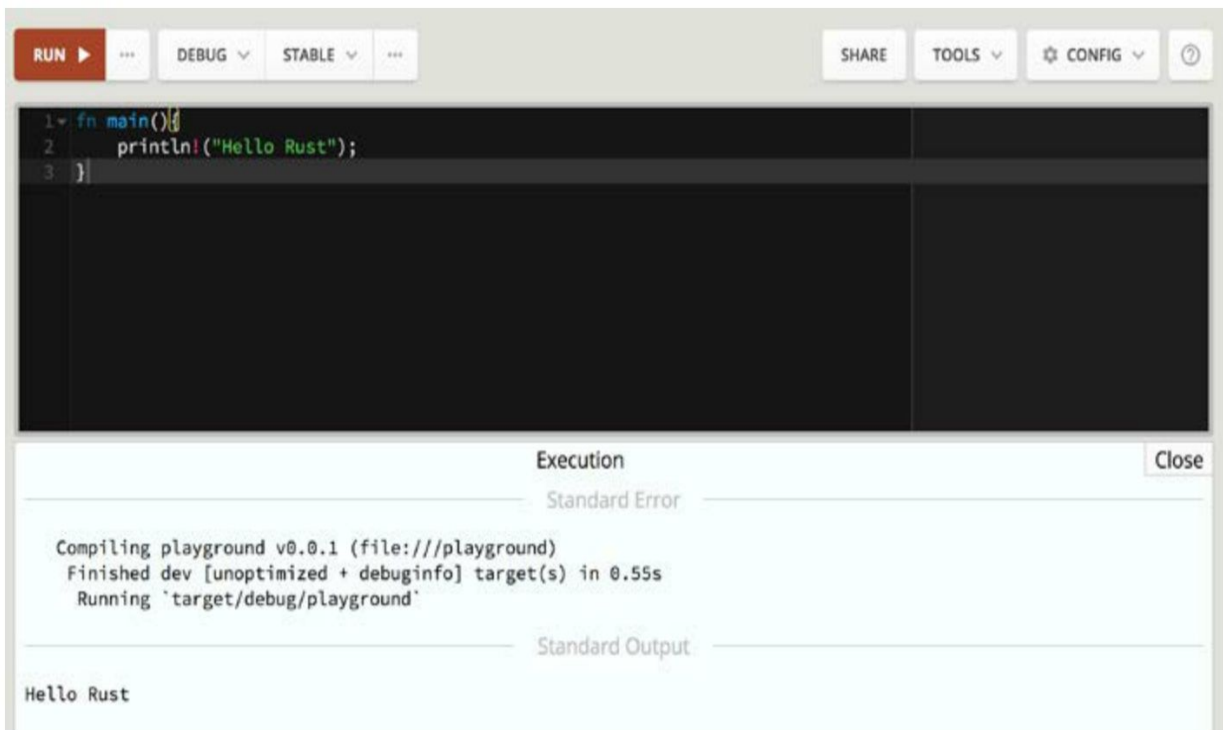
WebAssembly Rust Web Rust

-
- [1] rust-sgx-sdk
 - [2] triple <https://forge.rust-lang.org/platform-support.html>.
 - [3] <https://doc.rust-lang.org/reference/items/external-blocks.html>.
 - [4] <https://crates.io/crates/cc>.
 - [5] Linux macOS
 - [6] <https://github.com/rust-lang-nursery/rust-bindgen>.
 - [7] <https://github.com/eqrion/cbindgen>.
 - [8] <https://github.com/alexcrichton/ctest>.
 - [9] <https://github.com/TimNN/cargo-lipo>.
 - [10] <https://github.com/prevoty/jni-rs>.
 - [11] macOS .dylib
 - [12] Node.js V10.7.0 N-API
 - [13] <https://github.com/WebAssembly/reference-types/blob/master/proposals/reference-types/Overview.md>.
 - [14] <https://github.com/WebAssembly/host-bindings/blob/master/proposals/host-bindings/Overview.md>.
 - [15] <https://webassembly.studio/>.
 - [16] <https://webassembly.org/docs/semantics/>
 - [17] <https://webassembly.studio/?f=asqnsI6ru3o>
 - [18] <https://webassembly.studio/?f=ottwfve7all>
 - [19] Firefox Nightly
 - [20] <https://rustwasm.github.io/wasm-bindgen/introduction.html>.

□□A Rust□□□□□□

A.1 Rust

`rust-lang` Rust Playground
<https://play.rust-lang.org/?text=A-1>

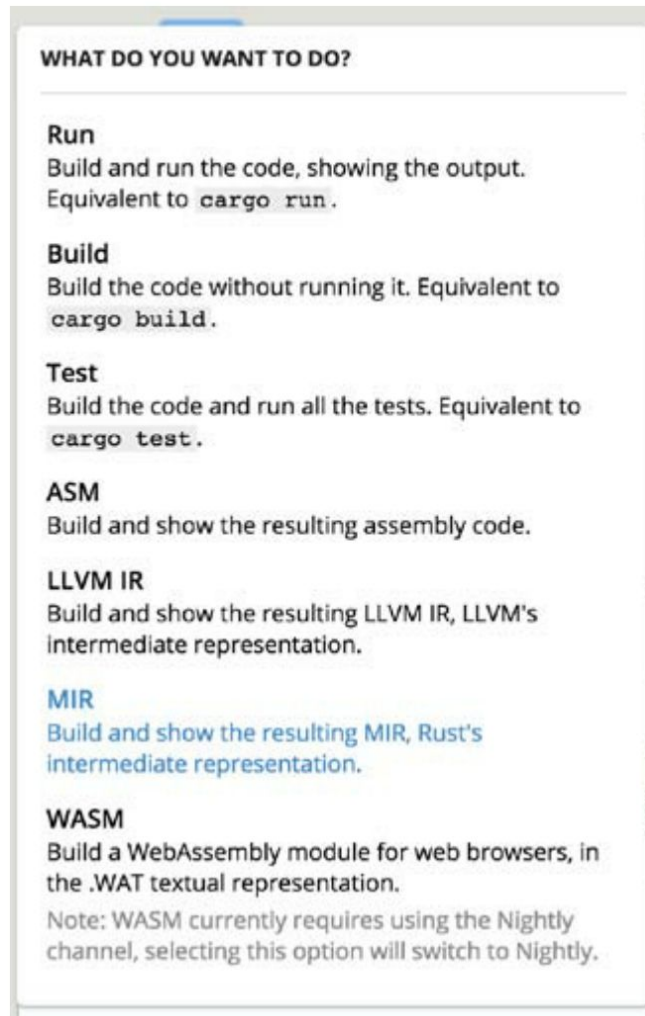


A-1 Playground

```
Rust Read-eval-print-loop REPL
Playground REPL
```

PlayGroud → ASM → LLVM IR → MIR → A-2

MIR MIR PlayGroud Rust
Stable Beat Nightly Debug Release



A-2 Rustのインストール

A.2 Rustのインストール

Rustのインストールには、rustcとcargoが必要です。

- ・ rustcはRustのコンパイラ

- ・ cargoはRustのパッケージ管理ツール

Rustのインストールには、Nightly、Beta、Stableの3つのチャンネルがあります。

- ・ Nightlyは最新のRustのビルドです。Nightlyのビルドは、Rustの最新の機能や、Rustの最新の機能のテスト版を含みます。

- ・ Betaは最新のRustのビルドです。Betaのビルドは、Rustの最新の機能や、Rustの最新の機能のテスト版を含みます。

- ・ Stableは最新のRustのビルドです。Stableのビルドは、Rustの最新の機能や、Rustの最新の機能のテスト版を含みます。

安装 Stable 或 Nightly 版本
安装 Nightly

A.2.1 Rust

Rust 安装 rustup Ruby rbnb Python
pyenv Node nvm

安装 rustup

```
curl https://sh.rustup.rs -sSf | sh
```

安装 Nightly

```
curl https://sh.rustup.rs -sSf | sh -s -- --default-toolchain nightly -y
```

Windows Mac Ubuntu
rustup Cargo rustc cargo rustup
UNIX \$HOME/.cargo/bin Windows
%USERPROFILE%\cargo\bin

安装 rustc

```
rustc --version
```

安装 rust

rustup rustup default
rustc

```
rustc default nightly
```

```
rustc default nightly-2018-05-12
```

rustup rustup-h rustup

A.2.2

Rustup USTC Rustup

1

```
export RUSTUP_DIST_SERVER=http://mirrors.ustc.edu.cn/rust-static  
export RUSTUP_UPDATE_ROOT=http://mirrors.ustc.edu.cn/rust-static/rustup
```

```

2 cargo
CARGO_HOME/.cargoconfig
[source.crates-io]
registry = "https://github.com/rust-lang/crates.io-index"
replace-with = 'ustc'
[source.ustc]
registry = "http://mirrors.ustc.edu.cn/crates.io-index"

```

A.3 Docker Rust

Dockerfile

```

FROM phusion/baseimage
ENV RUSTUP_HOME=/rust
ENV CARGO_HOME=/cargo
ENV PATH=/cargo/bin:/rust/bin:$PATH
RUN curl https://sh.rustup.rs -sSf | sh -s -- --default-toolchain nightly
-y

```

Nightly nightly stable
 nightly

```

RUN rustup default nightly-2018-05-12

```

A.4 Rust IDE

IDE Visual Studio Code IntelliJ IDEA
 Emacs Emacspace Vim Atom

A.5

A.5.1 Racer

Racer Rust Interllij IDEA Rust
 Racer AST

```

cargo install racer

```

rustup

```

rustup component add rust-src

```

rustup-init

```
export RUST_SRC_PATH="$(rustc --print sysroot)/lib/rustlib/src/rust/src"
```

A.5.2 RLS

RLS (Rust Language Server) 是一个为 IDE 提供 Rust 语言服务的工具。它允许 IDE 通过 RLS 与 Rust 编译器交互，从而提供智能感知、代码补全、错误检查等功能。IDE 可以通过 RLS 与 Rust 编译器交互。

RLS (Rust) 可以在 Visual Studio Code 中使用。安装 RLS 的方法如下：
1. 安装 nightly Rust 版本。

RLS 的 README [1] 提供了 rustup 和 racer 的安装说明。

A.5.3 cargo

Rust 的 cargo 是一个包管理器。它用于构建和管理 Rust 项目。cargo 提供了许多有用的命令，如 build、run、test 等。

clippy

Code Smell 是一个 Rust 工具，用于检测代码中的潜在问题。它可以通过 rustup 安装。

```
rustup component add clippy
```

rustfmt

rustfmt 是一个 Rust 工具，用于格式化代码。它可以通过 cargo 安装。

```
rustup component add rustfmt
```

cargo fix

1.29 版本 Cargo 引入了 cargo fix 命令，用于自动修复代码中的问题。它可以通过 cargo 安装。

[1] <https://github.com/rust-lang-nursery/rls#setup>.

00B Rust000000

000000Rust000000000000Rust000000Rust00020180900
000000000000CVE-2018-1000657 [1] 0Rust000GitHub0000
issues [2] 000000

0000000000

- 000VecDeque0T0000“00”000“00”00000UB
- Rust00segfault000000000000UB
- Rust000UB000000Unsafe Rust000
- 00UB000000000000000000std00ptr00write0000000000
000000000000
- Rust0000000000000000000000VecDeque000000000000
unsafe000
Segfault

000000**LLDB** 000issues000000000000000000000000LLDB
macOS0000000000Linux000GDB000000

B.1 0000

000000 **rustup install 1.20.0** 0000000000 Rust 0000000000
rustup default 1.20.0000000Rust000000

000000000000**VSCode** 000000**CodeLLDB** 000Mac000
Linux00GDB0000000000000000**cargo new lldb_demo** 000
src/main.rs 000000issues0000000000

00000000

```
1. use std::fmt;
2. use std::collections::VecDeque;
3. pub struct Packet
4. {
5.     pub payload: VecDeque<u8>,
6. }
```

```

7.  pub struct Header
8.  {
9.      pub data: Vec<u8>,
10. }
11. // 省略
12. impl Header
13. {
14.     pub fn new() -> Self
15.     {
16.         let data = Vec::with_capacity(20);
17.         Header{data}
18.     }
19. // 省略
20. }
21. fn push_ipv4(packet: &mut Packet)
22. {
23.     // 省略
24. }
25. fn push_mac(packet: &mut Packet)
26. {
27.     let mut header = Header::with_capacity(30);
28.     // 省略
29.     println!("new packet = {:?}", packet);
30. }
31. fn main()
32. {
33.     let mut packet = Packet::new();
34.     push_ipv4(&mut packet);
35.     push_mac(&mut packet);
36. }

```

在src/appendix/lldb.rs中添加lldb配置
 安装rust-lldb并配置VSCode B-1 VSCode Debug
 Add Configuration...


```

147 fn main()
148 {
149     let mut packet = Packet::new();
150     push_ipv4(&mut packet);
151     push_mac(&mut packet);
152 }

```

B-3 main

Debug

B-4 Step Over F10

```

138
139 let fcs = [0xD9, 0x58, 0xFB, 0xA8];
140 println!("old packet = {:?}", packet);
141 println!("pushing {:X} {:X} {:X} {:X} ", fcs[0], fcs[1], fcs[2], fcs[3]);
142 packet.push_back_bytes(&fcs);
143
144 println!("new packet = {:?}", packet);
145 }
146
147 fn main()
148 {
149     let mut packet = Packet::new();
150     push_ipv4(&mut packet);
151     push_mac(&mut packet);
152 }

```

B-4 Step Over F10

main B-5


```

0000000000000000 Bug 00000000000000000000000000000000
0000000000000000 rt lang_start 000000000000000000
0000000000000000 CALL STACK 00 std rt lang_start 00
0000000000000000 B-7 000

```

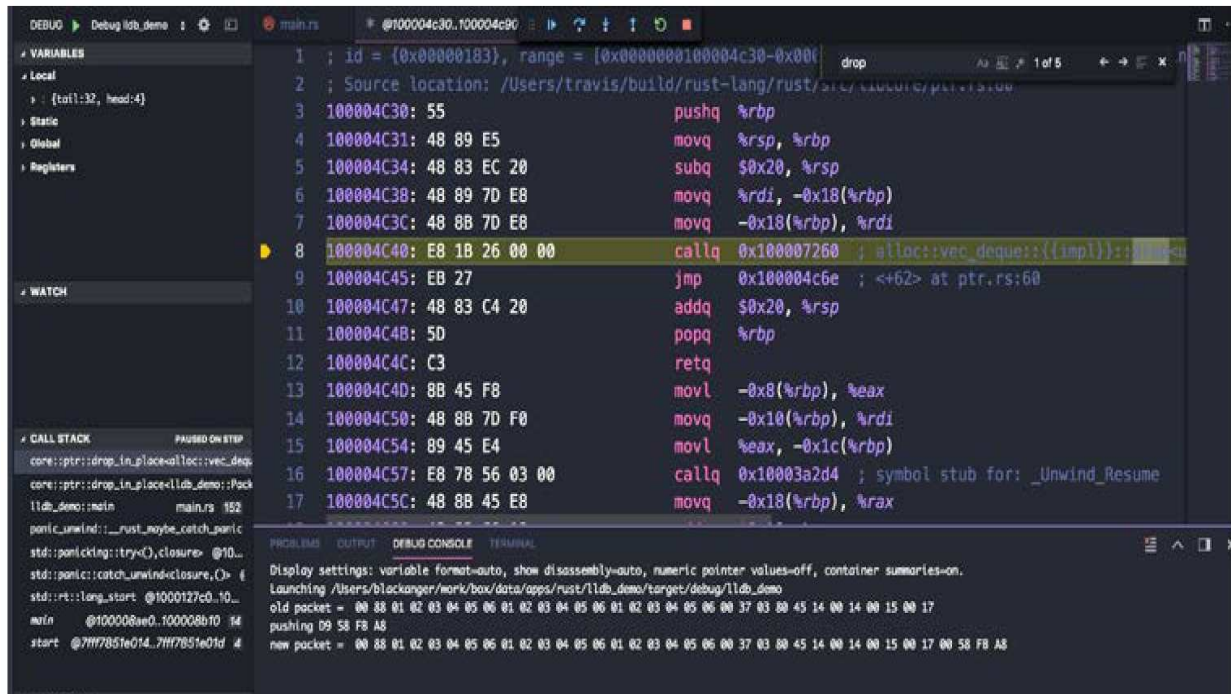
CALL STACKstdrtlang_start

The screenshot shows the LLDB debugger interface with the following components:

- Debugger Title Bar:** Shows the current session as "DEBUG: lladb_demo" and the loaded binary as "@100004ce0.100004c00".
- VARIABLES Panel:**
 - Local:** Shows a variable `drop` with a value of `0x000000000100004ce0-0x000000000100004c00`.
 - Static:** Empty.
 - Global:** Empty.
 - Registers:** Empty.
- WATCH Panel:** Empty.
- CALL STACK Panel:**
 - Paused on step: `core::ptr::drop_in_place<lladb_demo::Pack>`
 - Stack trace:
 - `lladb_demo::main main.rs:162`
 - `panic_unwind::__rust_maybe_catch_panic`
 - `std::panicking::try<O>, closure> @10...`
 - `std::panicking::catch_unwind<closure>, O> @...`
 - `std::rt::lang_start @1000127c0.10...`
 - `main @1000008ae0.1000008b10 34`
 - `start @7fff785fe014..7fff785fe01d 4`
- Disassembly Window:**
 - Address range: `0x000000000100004ce0-0x000000000100004c00`
 - Instructions:
 - `1 ; id = {0x00000018b}, range = {0x000000000100004ce0-0x000000000100004c00}`
 - `2 ; Source location: /Users/travis/build/rust-lang/rust/...`
 - `3 100004CE0: 55 pushq %rbp`
 - `4 100004CE1: 48 89 E5 movq %rsp, %rbp`
 - `5 100004CE4: 48 83 EC 10 subq $0x10, %rsp`
 - `6 100004CE8: 48 89 7D F8 movq %rdi, -0x8(%rbp)`
 - 7 100004CEC: 48 8B 7D F8 movq -0x8(%rbp), %rdi**
 - `8 100004CF0: E8 3B FF FF FF callq 0x100004c30 ; core::ptr::drop_in_place<alloc::ve...`
 - `9 100004CF5: 48 83 C4 10 addq $0x10, %rsp`
 - `10 100004CF9: 5D popq %rbp`
 - `11 100004CFA: C3 retq`
 - `12 100004CFB: 0F 1F 44 00 00 nopl (%rax,%rax)`
- PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL:**
 - Display settings: `variable-format=auto, show-disassembly=auto, numeric-pointer-values=off, container-summaries=on.`
 - Launching: `/Users/blackanger/work/box/data/apps/rust/lladb_demo/target/debug/lladb_demo`
 - old packet = `00 88 01 02 03 04 05 06 01 02 03 04 05 06 01 02 03 04 05 06 00 37 03 80 45 14 00 14 00 15 00 17`
 - pushing `D9 58 F8 A8`
 - new packet = `00 88 01 02 03 04 05 06 01 02 03 04 05 06 01 02 03 04 05 06 00 37 03 80 45 14 00 14 00 15 00 17 00 58 F8 A8`

□B-8□□□Step Over□F10□□□□□□□□□

Step Into F11 CALLSTACK
core ptr drop_in_place VecDeque
Step Into core ptr drop_in_place
B-9

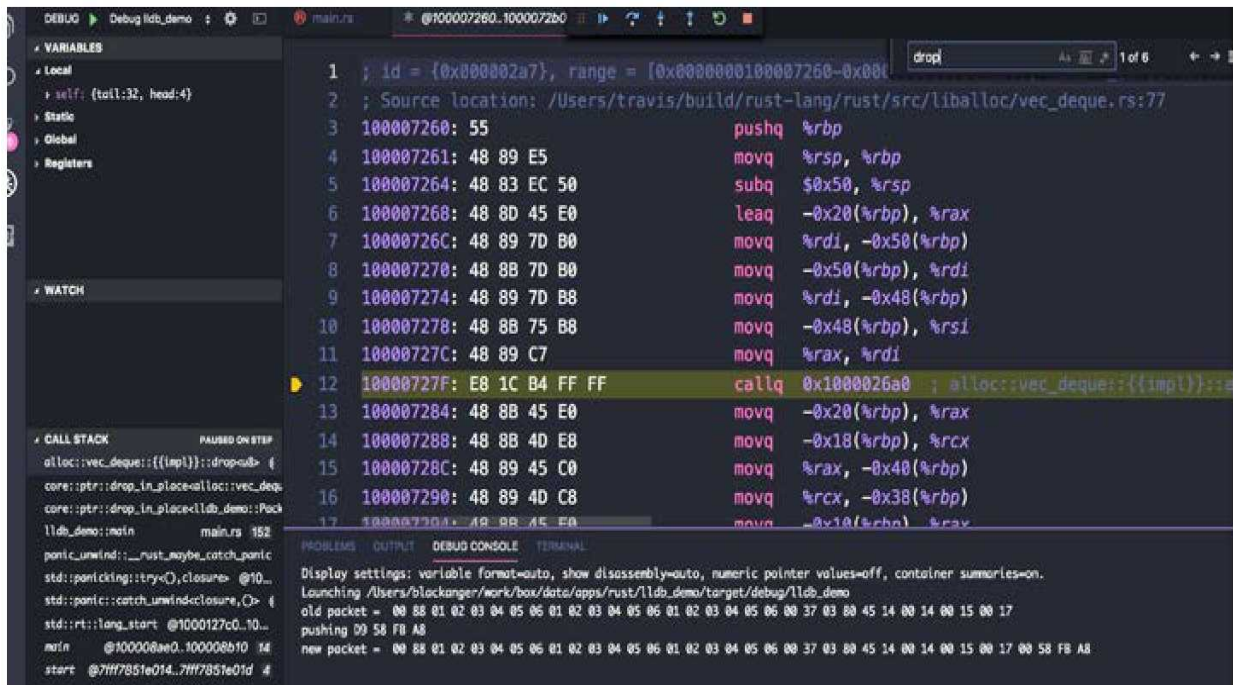


B-9 Step Into F11 drop_in_place

VecDeque

1. // VecDeque<T>析构函数
2. unsafe impl<#[may_dangle] T> Drop for VecDeque<T> {
3. fn drop(&mut self) {
4. let (front, back) = self.as_mut_slices();
5. unsafe {
6. // 调用 [T] 的析构函数
7. ptr::drop_in_place(front);
8. ptr::drop_in_place(back);
9. }
10. // RawVec 处理内存释放
11. }
12. }

VecDeque
Step IntoVecDequedropB-10

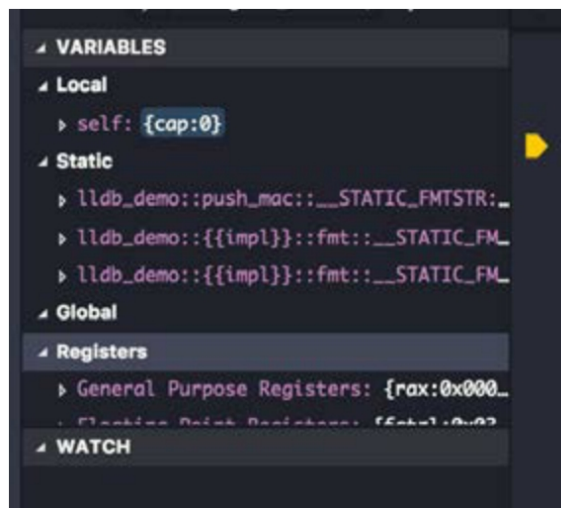


```
1 ; id = {0x000002a7}, range = [0x00000000100007260-0x00000000100007260]
2 ; Source location: /Users/travis/build/rust-lang/rust/src/liballoc/vec_deque.rs:77
3 100007260: 55                pushq %rbp
4 100007261: 48 89 E5          movq %rsp, %rbp
5 100007264: 48 83 EC 50       subq $0x50, %rsp
6 100007268: 48 8D 45 E0       leaq -0x20(%rbp), %rax
7 10000726C: 48 89 7D B0       movq %rdi, -0x50(%rbp)
8 100007270: 48 8B 7D B0       movq %rdi, -0x50(%rbp), %rdi
9 100007274: 48 89 7D B8       movq %rdi, -0x48(%rbp)
10 100007278: 48 8B 75 B8       movq -0x48(%rbp), %rsi
11 10000727C: 48 89 C7          movq %rax, %rdi
12 10000727F: E8 1C B4 FF FF   callq 0x1000026a0 ; alloc::vec_deque::{{impl}}::drop
13 100007284: 48 8B 45 E0       movq -0x20(%rbp), %rax
14 100007288: 48 8B 4D E8       movq -0x18(%rbp), %rcx
15 10000728C: 48 89 45 C0       movq %rax, -0x40(%rbp)
16 100007290: 48 89 4D C8       movq %rcx, -0x38(%rbp)
17 100007294: 48 8B 45 E0       movq -0x20(%rbp), %rax
```

B-10Step IntoF11vec_dequedrop

Step Intodrop

mainmainRust
VARIABLESB-11



```
▲ VARIABLES
▲ Local
  ▶ self: {cap:0}
▲ Static
  ▶ lldb_demo::push_mac::__STATIC_FMTSTR:
  ▶ lldb_demo::{{impl}}::fmt::__STATIC_FM
  ▶ lldb_demo::{{impl}}::fmt::__STATIC_FM
▲ Global
▲ Registers
  ▶ General Purpose Registers: {rax:0x000_
  ▶ Floating Point Registers: {fcr:0x00
▲ WATCH
```

B-11Step IntoVARIABLES

VecDeque RawVec
dealloc_bufferB-12

The screenshot shows a debugger window with the following components:

- VARIABLES:** Local variables include `self: [cap:64]`, `elem_size: i`, and `ptr: ""`. Static variables include `lldb_demo::push_vec::__STATIC_FHTSTR: (C)`, `lldb_demo::{{impl}}::fvec::__STATIC_FHTSTRS_`, and `lldb_demo::{{impl}}::fvec::__STATIC_FHTSTR:`. Global variables are listed under the **Global** section.
- CALL STACK:** The stack is paused on step 40. The call stack includes:
 - `alloc::raw_vec::{{impl}}::dealloc_buffer<u8,al`
 - `alloc::raw_vec::{{impl}}::drop<u8,alloc::heap:`
 - `core::ptr::drop_in_place<alloc::raw_vec::RawVe`
 - `core::ptr::drop_in_place<alloc::vec_deque::Vec`
 - `lldb_demo::main`
- Assembly:** The assembly view shows instructions from address 1000039AF to 1000039F8. The instruction at 1000039DF is highlighted: `callq 0x100002030 ; core::option::{{impl}}`.
- PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL:** The terminal shows the following output:


```
Display settings: variable format=auto, show disassembly=auto, numeric pointer values=off, container summaries=on.
Launching /Users/blockanger/work/box/data/apps/rust/lldb_demo/target/debug/lldb_demo
old packet = 00 88 01 02 03 04 05 06 01 02 03 04 05 06 01 02 03 04 05 06 00 37 03 00 45 14 00 14 00 15 00 17
pushing D9 58 F8 A8
new packet = 00 88 01 02 03 04 05 06 01 02 03 04 05 06 01 02 03 04 05 06 00 37 03 00 45 14 00 14 00 15 00 17 00 58 F8 A8
```

B-12 Step Into dealloc_buffer

heapdeallocB-13

The screenshot shows the **CALL STACK** window, which is paused on step. The call stack includes:

- `alloc::heap::{{impl}}::dealloc @100006d...`
- `alloc::raw_vec::{{impl}}::dealloc_buffer<u8,al`
- `alloc::raw_vec::{{impl}}::drop<u8,alloc::heap:`
- `core::ptr::drop_in_place<alloc::raw_vec::RawVe`
- `core::ptr::drop_in_place<alloc::vec_deque::Vec`

B-13 Step Into heapdealloc

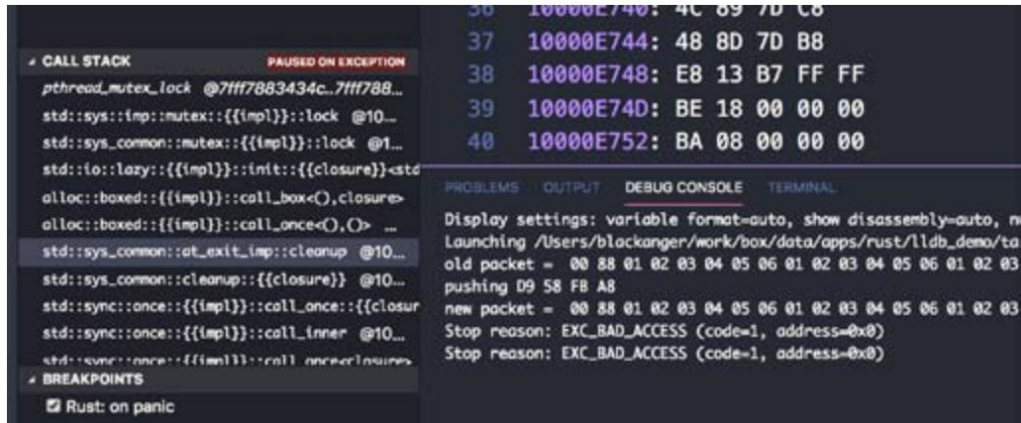
jemallocB-14

The screenshot shows the **CALL STACK** window, which is paused on step. The call stack includes:

- `alloc_jemalloc::contents::__rde_dealloc @...`
- `alloc::heap::{{impl}}::dealloc @100006d...`
- `alloc::raw_vec::{{impl}}::dealloc_buffer<u8,al`
- `alloc::raw_vec::{{impl}}::drop<u8,alloc::heap:`
- `core::ptr::drop_in_place<alloc::raw_vec::RawVe`

B-14 Step Into jemallocdealloc

Rust 1.20 RustJemallocdeallocJemalloc
 VSCodeB-15



B-15

stdsyscommonat_exit_impcleanup
 stdsyscommonat_exit_imp
 rt

VSCode LLDB DebugEXC_BAD_ACCESS
 pthread_mutex_lockpthread_mutex_lock
 libcAPIEXC_BAD_ACCESS
 “”

B.3

1 LLDB

2

3 main
 panic

4 main
 cleanupEXC_BAD_ACCESS

5 pthread_mutex_lockRust

[\[1\] https://cve.mitre.org/cgi-bin/cvename.cgi?name=%20CVE-2018-1000657.](https://cve.mitre.org/cgi-bin/cvename.cgi?name=%20CVE-2018-1000657)

[\[2\] https://github.com/rust-lang/rust/issues/44800.](https://github.com/rust-lang/rust/issues/44800)